Catalogue of architectural patterns characterized by constraint components, Version 1.0

Tu Minh Ton That, Chouki Tibermacine, Salah Sadou

Catalogue of architectural patterns characterized by constraint components, Version 1.0

Technical Report
IRISA/ArchWare-2013-TR-01

Minh Tu Ton That[1], Chouki Tibermacine[2] and Salah Sadou[1]

[1] IRISA, University of South Brittany, France
{minh-tu.ton-that, salah.sadou)@irisa.fr

[2] LIRMM, CNRS and Montpellier II University, France
tibermacin@lirmm.fr

Abstract. This report documents a catalogue of architectural patterns built from constraint components. Constraint component is a concept used to represent architectural constraints by components. It facilitates the reusability, the composability and the customizability of architectural constraints.
This report revises a list of existing architectural patterns in the literature and represents them using constraint components.

1 Introduction

This report revises a list of existing architectural patterns in the literature and represents them using constraint components. Architectural patterns are described in terms of unit constraints that assure their aimed characteristics. These unit constraints can be reused to compose different architectural patterns. Thus, for each revised pattern, we show its constituent unit constraints and for each constraint, we show the patterns in which it is reused.

This document is organized as follows: Section 2 introduces the CLACS profile in which constraints are written, Section 3 describes the list of architectural patterns in form of constraints, Section 4 goes through the list of constraint components that are used to compose patterns, Section 6 presents the list of patterns using constraint components.

2 CLACS profile

Component-Based Specification of Architectural Constraints (CLACS) UML Profile is dedicated to represent architectural constraints using the notion of component. In this section we only focus on a part of CLACS Profile that is used to represent the component structure of the architecture. In CLACS, a *component* is an instance of a *component descriptor*. A component has *ports* which are interaction points. Bindings between ports can be defined using *connectors*. As in **Fig. 1**, a Component Descriptor

extends the meta-class Component. A Component Instance extends the meta-class Property. This is due to the fact that in the UML Composite Structure Diagram, the internal structure of a composite component contains *Properties* typed by *Components*. A *Port* extends the meta-class Port. Finally, a Connector extends the meta-class *Connector*.

.

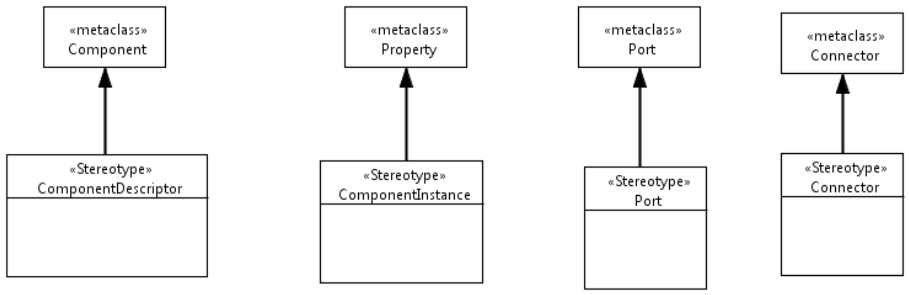


Fig. 1. Part of the CLACS profile representing the structural organization of components

For example, considering the following snippet of architectural constraint in OCL (Object Constraint Language):

```
context ComponentInstance

inv: self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->
size() >= 1
```

This constraint simply checks all internal component instances of a component and make sure that there are at least one of them. Notice that throughout this report, all the constraint components will be written based on CLACS Profile.

3 Pattern catalogue

This section will go through a list of architectural patterns in the literature. Each pattern will be described using textual architectural constraints. The following points should be noticed to be able to understand the list:

Remark 1. For certain patterns, there exist three different variants. The variant called *Context Independent* contains constraints that are applied to all components in the architecture. The variant called *Hybrid* contains constraints that are applied to only a part of architecture (in case that the architecture exposes many different patterns and we take care of only one of them). The variant called *Group* contains constraints with group granularity. In other words, instead of imposing constraints on components we impose constraints on groups of components.

Remark 2. Variants of patterns and their constituent constraints are numbered. The list of numbered constraints will be presented in Section 4.

Remark 3. We will not re-explain the structure of architectural patterns but directly describe their constituent constraints. For more details, the readers are invited to revisit the cited sources where patterns are described.

3.1 Pipes and Filters ([1][2][3])

3.1.1 (1) Variant 1 – Context Independent

(1) Each filter should be connected
(10) Pipes between each pair of filters should go into the same direction
(30) There is no cycle between filters

3.1.2 (2) Variant 2 – Hybrid

(2) In selected filters, each filter should be connected
(11) In selected filters, pipes between each pair of filters should go into the same direction
(31) In selected filters, there is no cycle between filters

3.1.3 (3) Variant 3 – Group

(3) Each group of filters should be connected
(12) Pipes between each pair of groups of filters should go into the same direction
(32) There is no cycle between groups of filters

3.2 Layers ([1][2][3])

3.2.1 (4) Variant 1 – Context Independent

(1) Each layer should be connected
(4) Each layer should be connected to at most 2 other layers
(7) The number of layers that are connected to only 1 other layer is equal to 2 (top and bottom layers)

3.2.2 (5) Variant 2 – Hybrid

(2) In selected layers, each layer should be connected
(5) In selected layers, each layer should be connected to at most 2 other layers
(8) In selected layers, the number of layers that are connected to only 1 other layer is equal to 2 (top and bottom layers)

3.2.3 (6) Variant 3 – Group

(3) Each group of layers should be connected
(6) Each group of layers should be connected to at most 2 other groups of layers
(9) The number of groups of layers that are connected to only 1 other group of layers is equal to 2 (top and bottom group of layers)

3.3 Layered Pipes And Filters

3.3.1 (7) Variant 1 – Context Independent

(1) Each filter should be connected

(4) Each filter should be connected to at most 2 other filters

(7) The number of filters that are connected to only 1 other filter is equal to 2 (left-most and right-most filters)

(10) Pipes between each pair of filters should go into the same direction

(13) There is exactly 1 filter having only successors

(16) There is exactly 1 filter having only predecessors

3.3.2 (8) Variant 2 – Hybrid

(2) In selected filters, each filter should be connected

(5) In selected filters, each filter should be connected to at most 2 other filters

(8) In selected filters, the number of filters that are connected to only 1 other filter is equal to 2 (left-most and right-most filters)

(11) In selected filters, pipes between each pair of filters should go into the same direction

(14) In selected filters, there is exactly 1 filter having only successors

(17) In selected filters, there is exactly 1 filter having only predecessors

3.3.3 (9) Variant 3 – Group

(3) Each group of filters should be connected

(6) Each group of filters should be connected to at most 2 other groups of filters

(9) The number of groups of filters that are connected to only 1 other group of filters is equal to 2 (left-most and right-most group of filters)

(12) Pipes between each pair of groups of filters should go into the same direction

(15) There is exactly 1 group of filters having only successors

(18) There is exactly 1 group of filters having only predecessors

3.4 (10) N-tier

(3) Each tier should be connected

(6) Each tier should be connected to at most 2 other tiers

(9) The number of tiers that are connected to only 1 other tier is equal to 2 (top and bottom tier)

(12) Connectors between each pair of tiers should go into the same direction

(15) There is exactly 1 tier having only successors

(18) There is exactly 1 tier having only predecessors

3.5 MVC (the variant that is equivalent to 3-layer pattern) ([1][2])

3.5.1 (11) Variant 1 – Context independent

(1) Each component should be connected

(7) The number of components that are connected to only 1 other component is equal to 2 (first (M) and last (C) component)

(59) There are exactly 3 component (M, V, C)

3.5.2 (12) Variant 2 – Hybrid

(2) In selected components, each component should be connected

(8) In selected components, the number of components that are connected to only 1 other component is equal to 2 (first (M) and last (C) component)

(60) In selected components, there are exactly 3 component (M, V, C)

3.5.3 (13) Variant 3 – Group

(3) Each group of components should be connected

(9) The number of groups of components that are connected to only 1 other group of components is equal to 2 (first (M) and last (C) group of components)

(19) There are exactly 3 groups of components (M, V, C)

3.6 (14) PAC ([1])

(3) Each layer should be connected

(6) Each layer should be connected to at most 2 other layers

(9) The number of layers that are connected to only 1 other layer is equal to 2 (top and bottom layers)

(33) Each component in an agent should be connected

(34) The number of components in an agent that are connected to only one other component is equal to two (first (M) and last (C) component)

(35) There are exactly 3 components in each agent (M, V, C)

(36) For every agent, there's only the Controller connected to the outside

3.7 Pipeline ([1])

3.7.1 (15) Variant 1 – Context Independent

(1) Each filter should be connected

(4) Each filter should be connected to at most 2 other filters

(7) The number of filters that are connected to only 1 other filter is equal to two (left-most and right-most filters)

(13) There is exactly 1 filter having only successors

(16) There is exactly 1 filter having only predecessors

(27) There's at most 1 pipe between each pair of filters

3.7.2 (16) Variant 2 – Hybrid

(2) In selected filters, each filter should be connected
(5) In selected filters, each filter should be connected to at most 2 other filters
(8) In selected filters, the number of filters that are connected to only 1 other filter is equal to 2 (left-most and right-most filters)
(14) In selected filters, there is exactly 1 filter having only successors
(17) In selected filters, there is exactly 1 filter having only predecessors
(28) In selected filters, there's at most 1 pipe between each pair of filters

3.7.3 (17) Variant 3 – Group

(3) Each group of filters should be connected
(6) Each group of filters should be connected to at most 2 other group of filters
(9) The number of groups of filters that are connected to only 1 other group of filters is equal to 2 (left-most and right-most group of filters)
(15) There is exactly 1 group of filters having only successors
(18) There is exactly 1 group of filters having only predecessors
(29) There's at most 1 pipe between each pair of group of filters

3.8 (18) Indirection Layer ([1])

(3) Each layer should be connected
(9) The number of layers that are connected to only 1 other layer is equal to 2 (left-most and right-most layers)
(12) Connectors between each pair of layers should go into the same direction
(15) There is exactly 1 layer having only successors
(18) There is exactly 1 layer having only predecessors
(19) There are exactly 3 layers
(20) The sub-system layer (3rd) has only predecessors
(21) Wrappers (in the 2nd layer) must wrap at least one component
(26) Wrappers (in the 2nd layer) do not communicate with each other

3.9 (19, 20, 21) Star (3 variants: Shared Repository pattern, Active Repository pattern, Black Board pattern) ([1])

(3) Each layer should be connected
(12) Connectors between each pair of layers should go into the same direction
(22) There are exactly 2 layers
(23) The sever layer (2nd) has only predecessors

3.10(22) Façade ([1])

(3) Each layer should be connected

(12) Connectors between each pair of layers should go into the same direction
(22) There are exactly 2 layers
(23) The 2nd layer has only predecessors (23)
(37) All components in the client layer (1st) which want to pass through sub-system layer (2nd) must pass through the facade component
(38) All predecessors of facade must be in the client layer (1st)
(39) All successors of facade must be in the sub system (2nd)
(40) Facade must wrap at least one component

3.11(23) Replicated component

(3) Each layer should be connected
(6) Each layer should be connected to at most two other layers
(9) The number of layers that are connected to only one other layer is equal to 2
(12) Connectors between each pair of layers should go into the same direction
(15) There is exactly 1 layer having only successors
(18) There is exactly 1 layer having only predecessors
(26) Components in the dispatcher layer (2nd) should not be connected to each other
(41) There are 5 layers
(42) The 5th layer has only predecessors
(43) Components in the replicated component layer (3rd) should not be connected to each other
(44) Components in the centralizer layer(4th) should not be connected to each other
(45) There are more than one replicated components in the 3rd layer
(46) Replicated components (3rd layer) should be connected to dispatchers and centralizers

3.12Ring

3.12.1 (24) Variant 1 – Context Independent

(1) Each component should be connected
(27) There's at most 1 connector between each pair of components
(47) Each component should be connected to exactly 2 other components
(50) There is exactly no component having only successors
(53) There is exactly no component having only predecessors

3.12.2 (25) Variant 2 – Hybrid

(2) In selected components, each component should be connected
(28) In selected components, there's at most 1 connector between each pair of components
(48) In selected components, each component should be connected to exactly 2 other components

(51) In selected components, there is exactly no component having only successors

(54) In selected components, there is exactly no component having only predecessors

3.12.3 (26) Variant 3 – Group

(3) Each group of components should be connected

(29) There's at most 1 connector between each pair of groups of components

(49) Each group of components should be connected to exactly 2 other groups of components

(52) There is exactly no group of components having only successors

(55) There is exactly no group of components having only predecessors

3.13(27) Legacy Wrapper ([4])

(3) Each layer should be connected

(9) The number of layers that are connected to only 1 other layer is equal to 2 (left-most and right-most layers)

(12) Connectors between each pair of layers should go into the same direction

(15) There is exactly 1 layer having only successors

(18) There is exactly 1 layer having only predecessors

(19) There are exactly 3 layers

(20) The 3rd layer has only predecessors

(24) There is exactly one legacy wrapper in the 2nd layer

3.14(28) Client-Server ([1][2][3])

(3) Each layer should be connected

(12) Connectors between each pair of layers should go into the same direction

(22) There are exactly 2 layers

(23) The sever layer (2nd) has only predecessors

(25) Components in the client layers (1st layer) should not be connected to each other

3.15(29) Microkernel ([1])

(3) Each layer should be connected

(6) Each layer should be connected to at most 2 other layers

(9) The number of layers that are connected to only 1 other layer is equal to 2 (left-most and right-most layers)

(12) Connectors between each pair of layers should go into the same direction

(15) There is exactly 1 layer having only successors

(18) There is exactly 1 layer having only predecessors

(56) There are exactly 4 layers

(57) The microkernel layer (3rd) has exactly 1 component (Microkernel)
(58) The internal server layer (4th) has only predecessors

3.16(30) Client-Server with Broker ([2])

(3) Each layer should be connected
(9) The number of layer that are connected to only 1 other layer is equal to 2 (left-most and right-most layers)
(12) Connectors between each pair of layers should go into the same direction
(15) There is exactly 1 layer having only successors
(18) There is exactly 1 layer having only predecessors
(19) There are exactly 3 layers
(20) The server layer (3rd) has only predecessors
(24) Brokers (in the 2nd layer) must wrap at least 1 components
(25) Components in the client layer (1st) should not be connected to each other

3.17(31) Bus

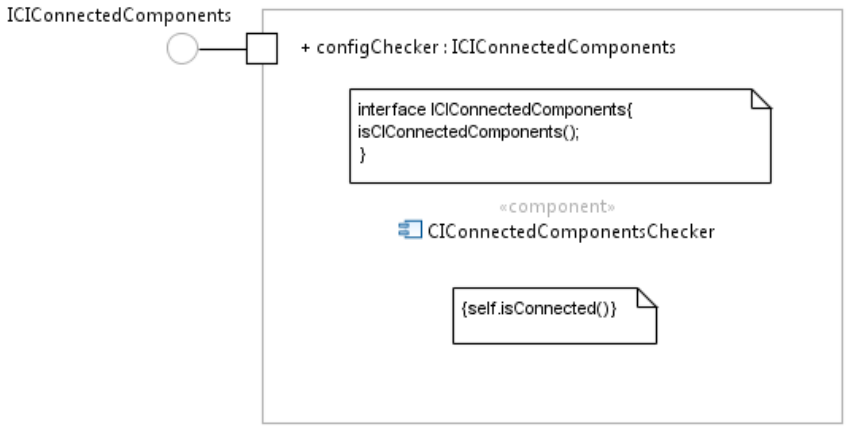
(3) Each layer should be connected
(9) The number of layer that are connected to only 1 other layer is equal to 2 (left-most and right-most layers)
(12) Connectors between each pair of layers should go into the same direction
(15) There is exactly 1 layer having only successors
(18) There is exactly 1 layer having only predecessors
(19) There are exactly 3 layers
(20) The server layer (3rd) has only predecessors
(24) There is exactly 1 bus (in the 2nd layer)
(25) Components in the client layer (1st) should not be connected to each other

4 List of constraint components

This section will go through a list of constraint components used to compose patterns. For each constraint component, “CI” at the beginning means Context Independent, “H” means Hybrid and “G” means Group

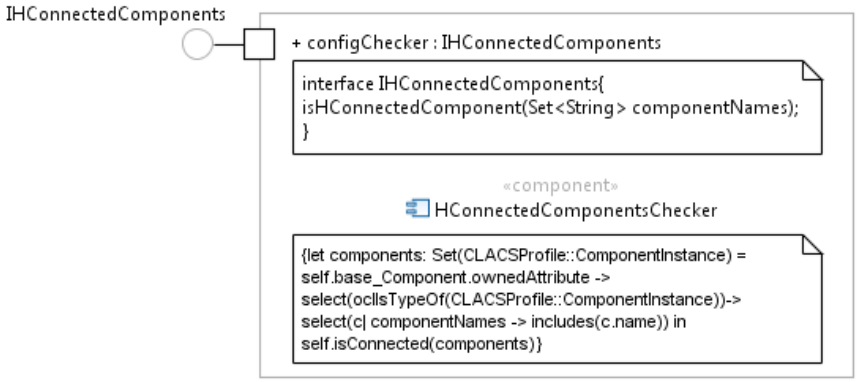
4.1 (1) CIConnectedComponentsChecker

Variant type: Context independent
Semantics: Each component should be connected.
Configuration:



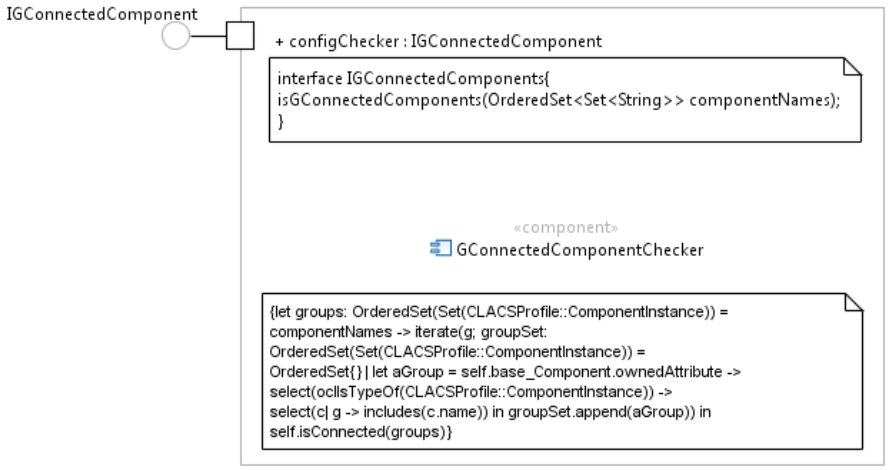
4.2 (2) HConnectedComponentsChecker

Variant type: Hybrid
Semantics: Each component in the selected components should be connected.
Configuration:



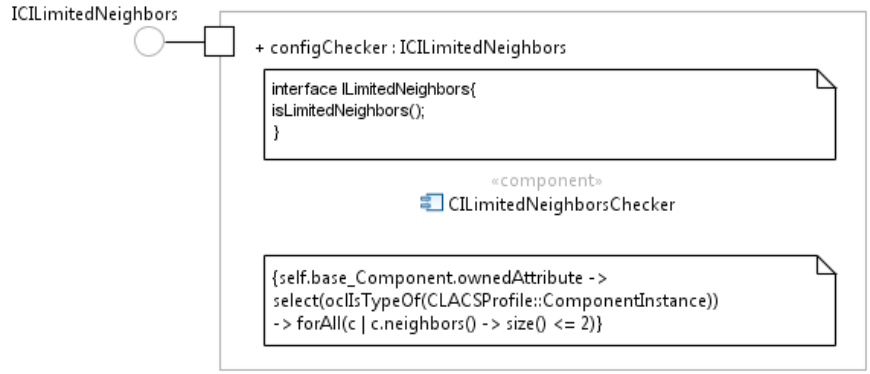
4.3 (3) GConnectedComponentsChecker

Variant type: Group
Semantics: Each group of components in the selected components should be connected.
Configuration:



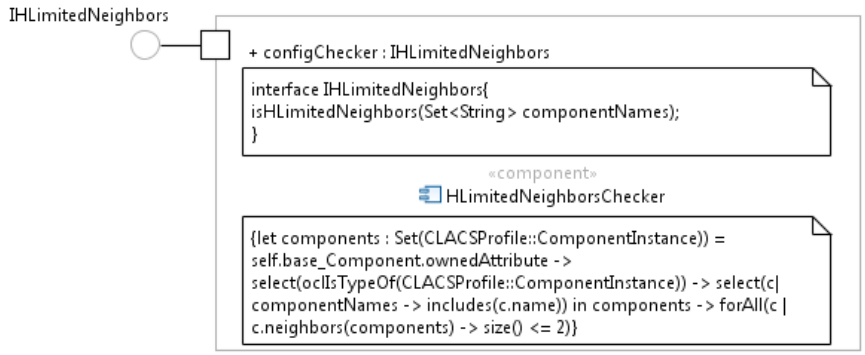
4.4 (4) CILimitedNeighborsChecker

Variant type: Context independent
Semantics: Each component should be connected to at most 2 other components.
Configuration:



4.5 (5) HLimitedNeighborsChecker

Variant type: Hybrid
Semantics: Each component in the selected components should be connected to at most 2 other components.
Configuration:

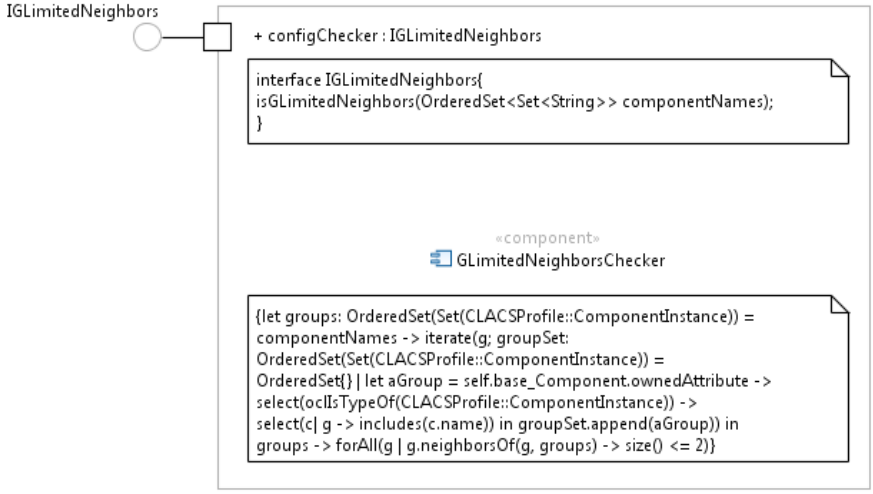


4.6 (6) GLimitedNeighborsChecker

Variant type: Group

Semantics: Each group of components in the selected components should be connected to at most 2 other groups.

Configuration:

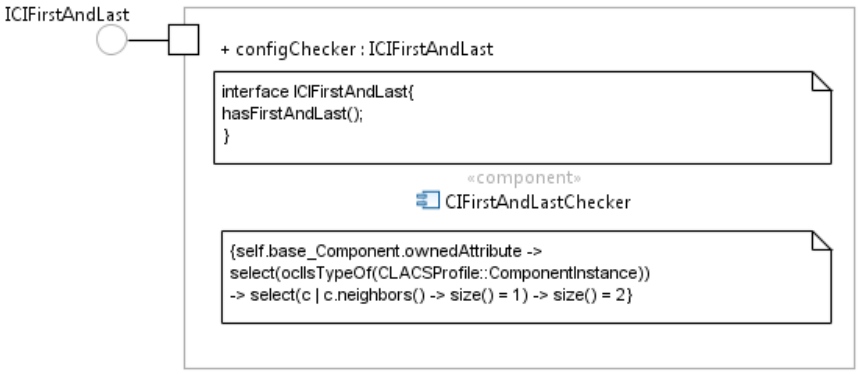


4.7 (7) CIFirstAndLastChecker

Variant type: Context independent

Semantics: The number of components that are connected to only one other component is equal to 2 (first and last components)

Configuration:

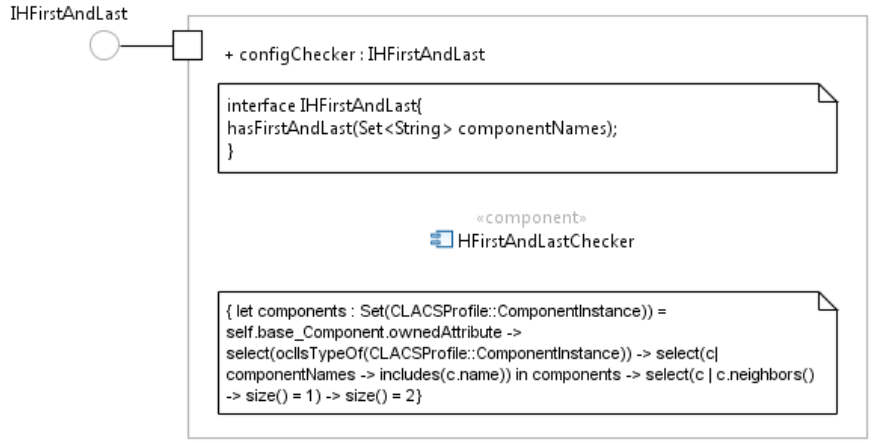


4.8 (8) HFirstAndLastChecker

Variant type: Hybrid

Semantics: The number of components in the selected components that are connected to only one other component is equal to 2 (first and last components)

Configuration:

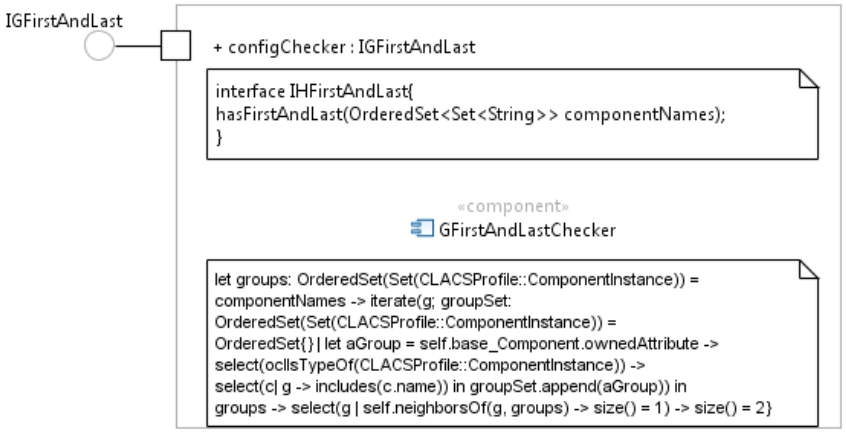


4.9 (9) GFirstAndLastChecker

Variant type: Group

Semantics: The number of group of components in the selected components that are connected to only one other group is equal to 2 (first and last groups)

Configuration:

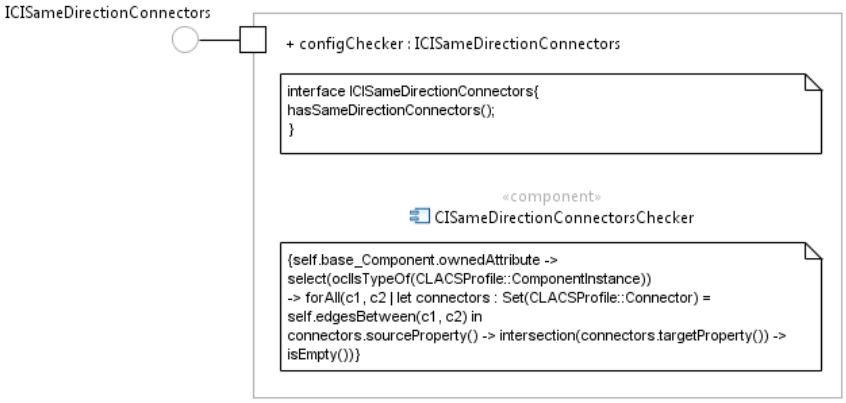


4.10 (10) CISameDirectionConnectorsChecker

Variant type: Context independent

Semantics: Connectors between each pair of components should go into the same direction

Configuration:

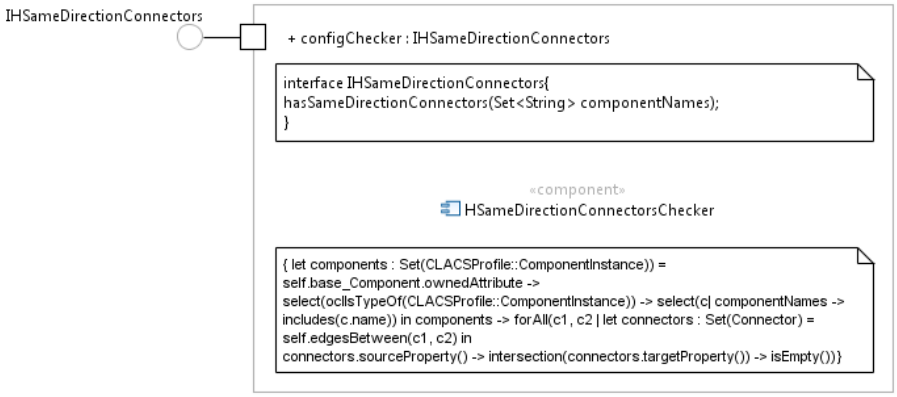


4.11 (11) HSameDirectionConnectorsChecker

Variant type: Hybrid

Semantics: Connectors between each pair of selected components should go into the same direction

Configuration:

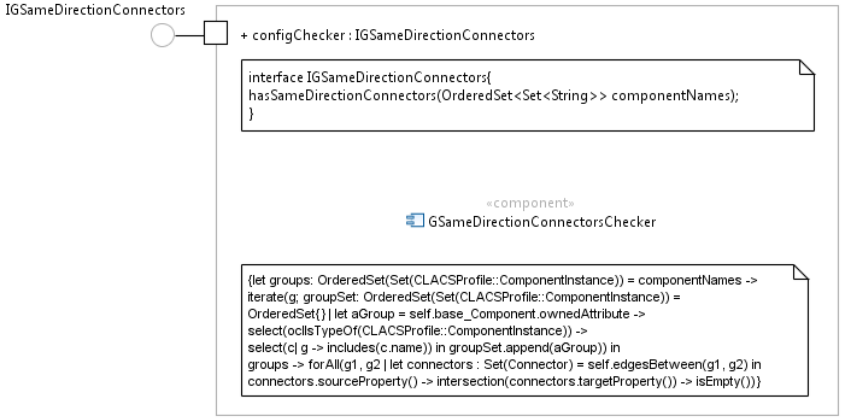


4.12 (12) GSameDirectionConnectorsChecker

Variant type: Group

Semantics: Connectors between each pair of groups of components should go into the same direction.

Configuration:

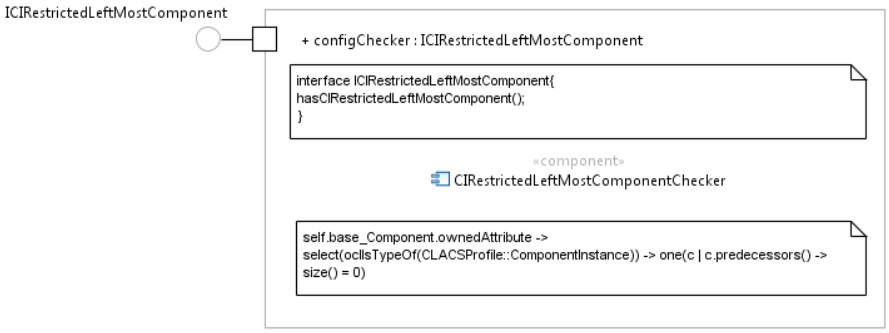


4.13 (13) CIRestrictedLeftMostComponentChecker

Variant type: Context independent

Semantics: There is exactly 1 component having only successors

Configuration:

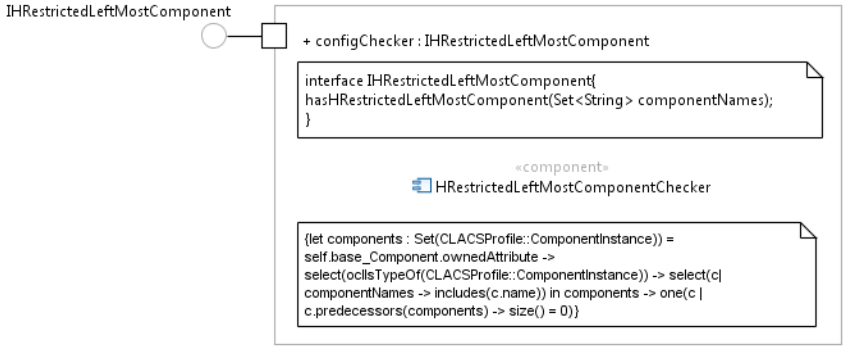


4.14 (14) HRestrictedLeftMostComponentChecker

Variant type: Hybrid

Semantics: There is exactly 1 component among selected components that has only successors.

Configuration:

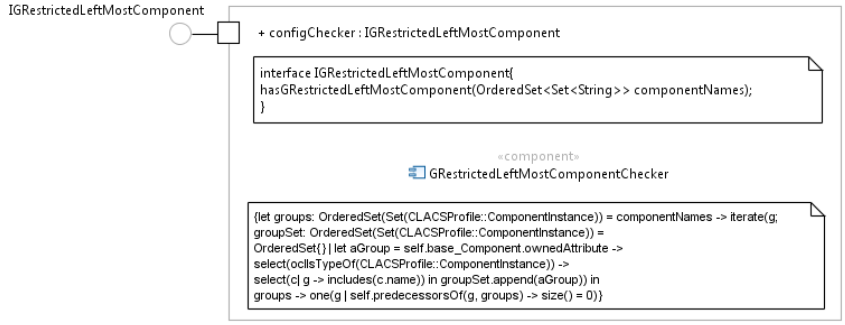


4.15 (15) GRestrictedLeftMostComponentChecker

Variant type: Group

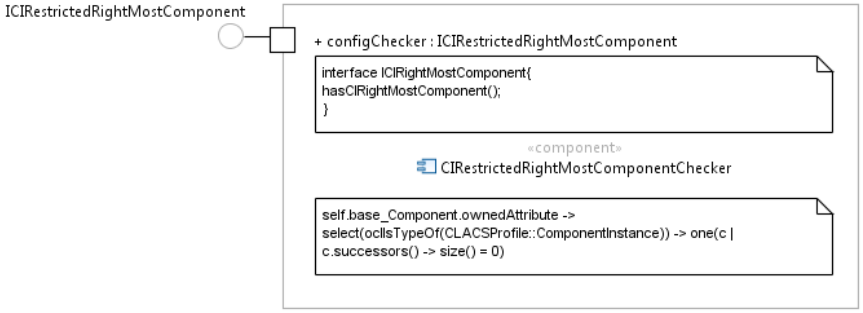
Semantics: There is exactly 1 group of components among selected groups that has only successors

Configuration:



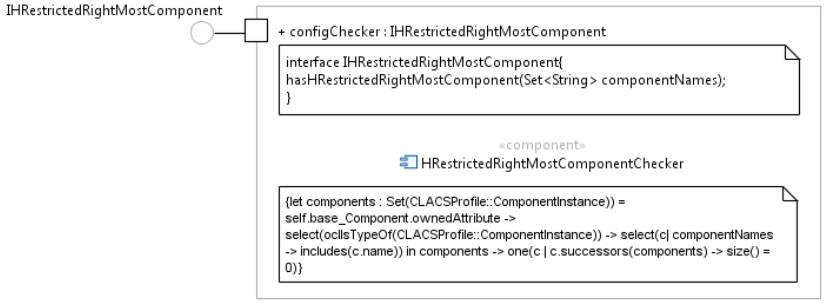
4.16 (16) CIRestrictedRightMostComponentChecker

Variant type: Context independent
Semantics: There is exactly 1 component having only predecessors.
Configuration:



4.17 (17) HRestrictedRightMostComponentChecker

Variant type: Hybrid
Semantics: There is exactly 1 component among selected components that has only predecessors.
Configuration:

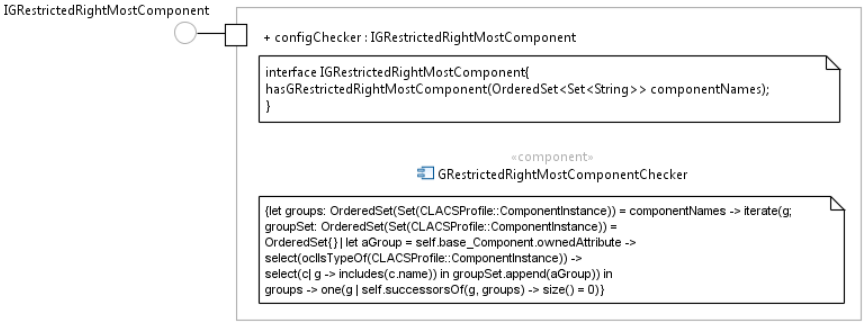


4.18 (18) GRestrictedRightMostComponentChecker

Variant type: Group

Semantics: There is exactly 1 group of components among selected groups that has only predecessors

Configuration:

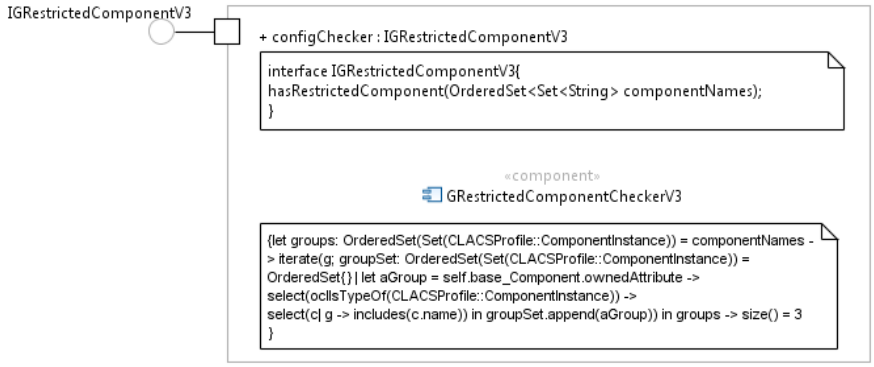


4.19 (19) GRestrictedComponentsCheckerV3

Variant type: Group

Semantics: There are exactly 3 components.

Configuration:

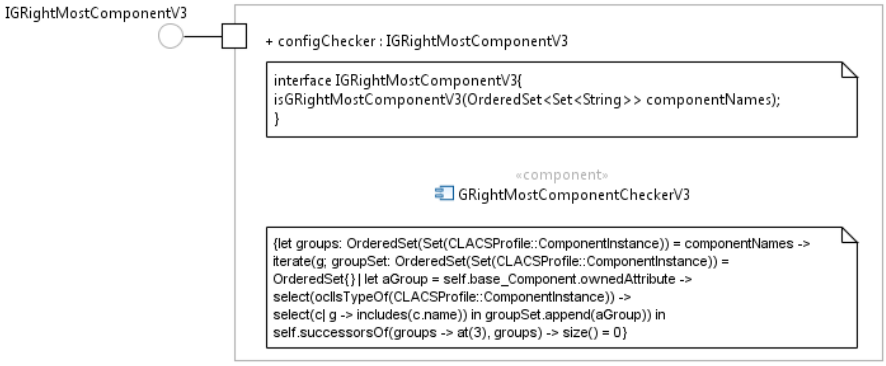


4.20 (20) GRightMostComponentCheckerV3

Variant type: Group

Semantics: The 3rd group of components among selected groups has only predecessors.

Configuration:

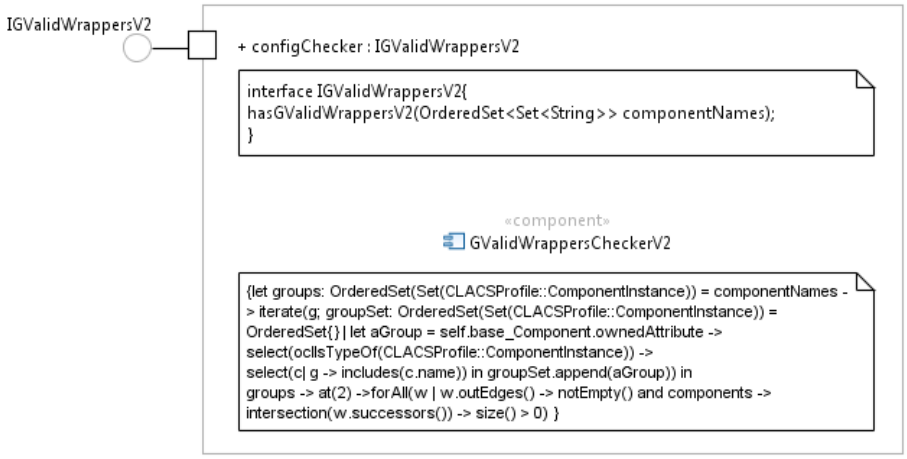


4.21 (21) GValidWrappersCheckerV2

Variant type: Group

Semantics: Every component (wrapper) in the 2nd group of components must have successors outside of its group.

Configuration:

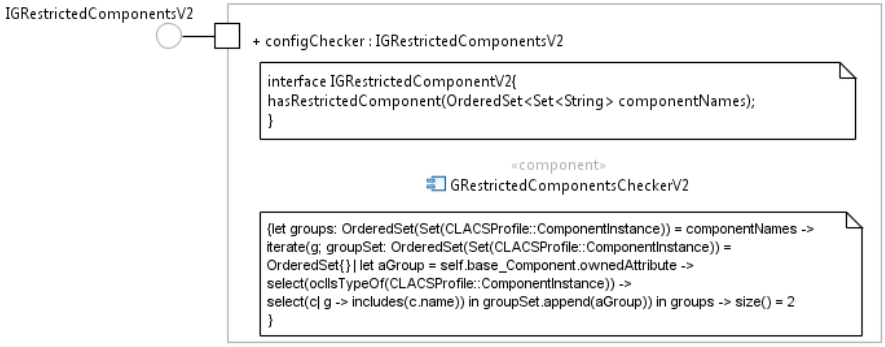


4.22 (22) GRestrictedComponentsCheckerV2

Variant type: Group

Semantics: There are exactly 2 groups of components.

Configuration:

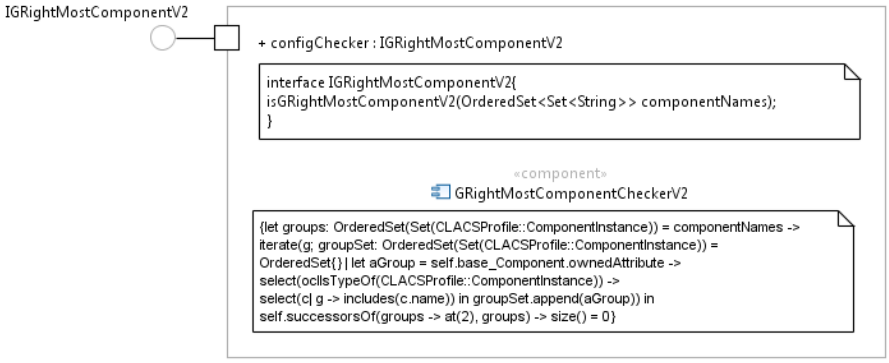


4.23 (23) GRightMostComponentCheckerV2

Variant type: Group

Semantics: The 2nd group of components among selected groups has only predecessors.

Configuration:

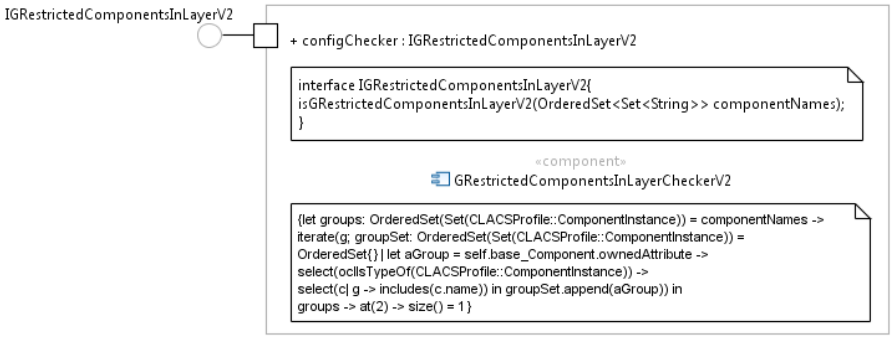


4.24 (24) GRestrictedComponentsInLayerCheckerV2

Variant type: Group

Semantics: The 2nd group of components among selected groups has only one component.

Configuration:

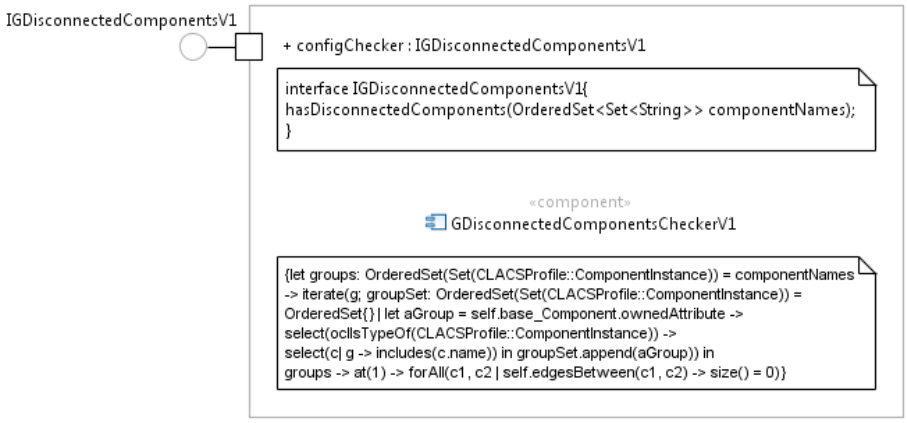


4.25 (25) GDisconnectedComponentsCheckerV1

Variant type: Group

Semantics: Every component in the 1st group of components must be disconnected to other components in the same group.

Configuration:

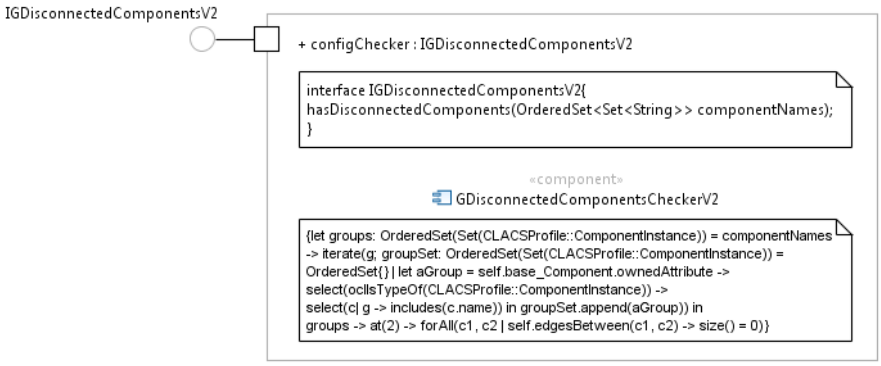


4.26 (26) GDisconnectedComponentsCheckerV2

Variant type: Group

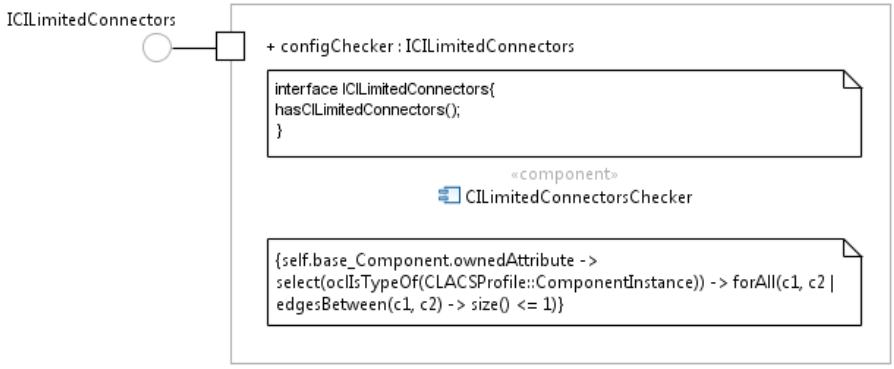
Semantics: Every component in the 2nd group of components must be disconnected to other components in the same group.

Configuration:



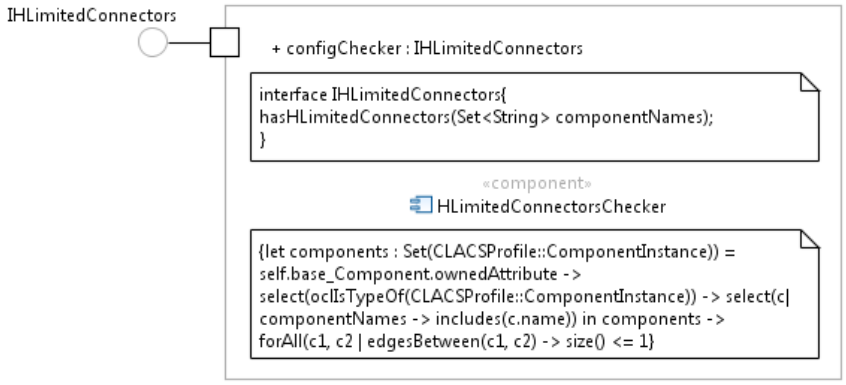
4.27 (27) CILimitedConnectorsChecker

Variant type: Context Independent
Semantics: There are at most 1 connector between each pair of components.
Configuration:



4.28 (28) HLimitedConnectorsChecker

Variant type: Hybrid
Semantics: There is at most 1 connector between each pair of selected components.
Configuration:

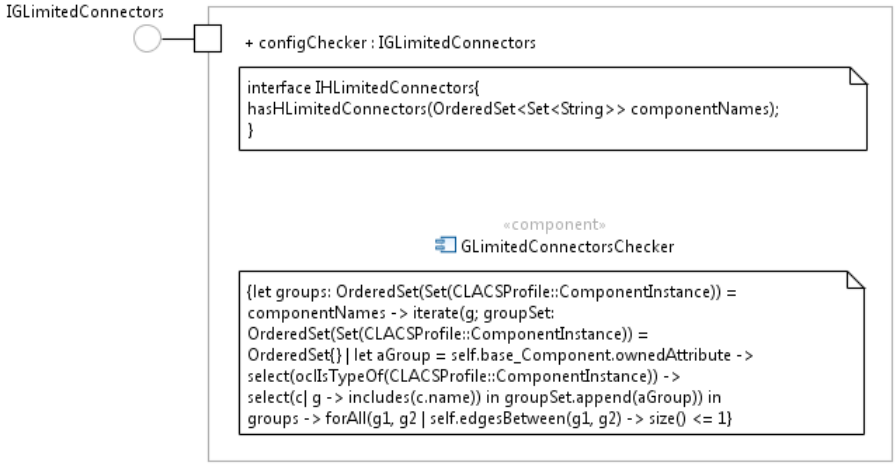


4.29 (29) GLimitedConnectorsChecker

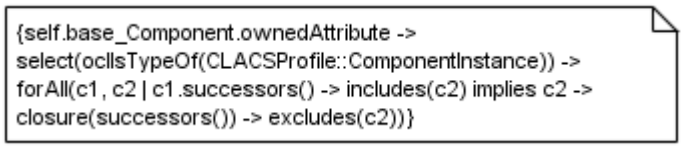
Variant type: Group

Semantics: There are at most 1 connector between each pair of group of components.

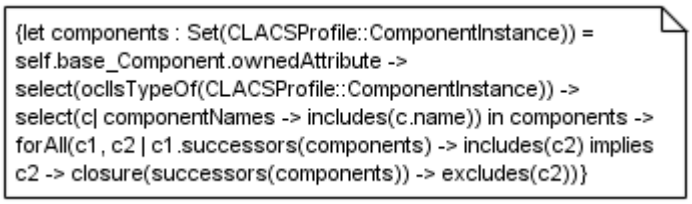
Configuration:



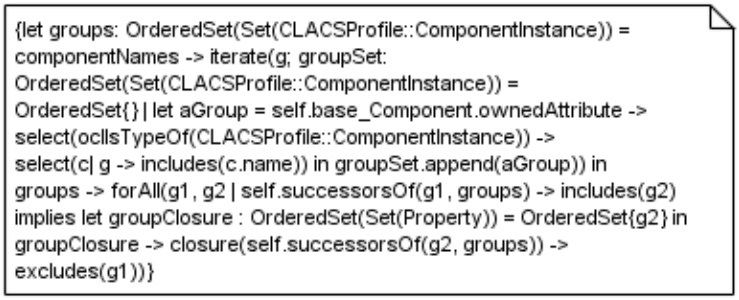
4.30 (30) in CIPipesAndFilters



4.31 (31) in HPipesAndFilters



4.32 (32) in GPipesAndFilters



4.33 (33, 34, 35, 36) in PAC

```
{let groups: OrderedSet(Set(CLACSProfile::ComponentInstance)) = componentNames -> iterate(g;
groupSet: OrderedSet(Set(CLACSProfile::ComponentInstance)) =
OrderedSet{} | let aGroup = self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->
select(c| g -> includes(c.name)) in groupSet.append(aGroup)) in
--Each component in an agent should be connected
groups -> forAll(g | g -> forAll( c | c.isConnected()))
and
--The number of components in an agent that are connected to only one other component is equal to 2
groups -> forAll(g | g -> forAll (c | c.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance))
-> forAll(c | c.neighbors() -> size() <= 2)))
and
--There are exactly 3 components in each agent
groups -> forAll(g | g -> forAll (c | c.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->size() = 3))
and
--For every agent, there is only the Controller (3rd component) connected to the outside
groups -> forAll(g | g -> forAll (c | c.base_Component.ownedConnector -> one(con | con.kind =
ConnectorKind::delegation and con.end.partWithPort ->any(p | p <>c).successors() -> size() = 0))))}
```

4.34 (37, 38, 39, 40) in GFaçade

```
{let groups: Set(Set(CLACSProfile::ComponentInstance)) = componentNames -> iterate(g; groupSet:
Set(Set(CLACSProfile::ComponentInstance)) =
Set{} | let aGroup = self.base_Component.ownedAttribute -> select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->
select(c| g -> includes(c.name)) in groupSet.append(aGroup)) in
let facade : CLACSProfile::ComponentInstance = self.base_Component.ownedAttribute -> any(name = facadeName) in
--All components in the client layer which want to pass through the sub-system layer must pass through the facade component
self.edgesBetween(clientComps, subSystemComps).targetProperty() -> size() = 1
and
self.edgesBetween(clientComps, subSystemComps).targetProperty() -> asSequence() -> first() = facade
and
--All successors of facade must be in the sub system
groups -> last() -> includesAll(facade.successors())
and
--All predecessors of facade must be in the client layer
groups -> first() -> includesAll(facade.predecessors())
and
--Facade must be connected to at least one component of the subsystem
facade.successors() -> size() > 0}
```

4.35 (41, 42, 43, 44, 45, 46) in GReplicated Component

```
{let groups: Set(Set(CLACSProfile::ComponentInstance)) = componentNames ->
iterate(g; groupSet: Set(Set(CLACSProfile::ComponentInstance)) =
Set{} | let aGroup = self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->
select(c| g -> includes(c.name)) in groupSet.append(aGroup)) in
-- There are 5 layers
groups -> size() = 5
and
-- The 5th layer has only predecessors
self.successorsOf(groups -> at(5), groups) -> size() = 0
and
-- Components in the 3rd layer (replicated component layer) are not connected to each
other
groups -> at(3) -> forAll(c1, c2 | self.edgesBetween(c1, c2) -> size() = 0)
and
-- Components in the 4th layer (centralizer layer) are not connected to each other
groups -> at(4) -> forAll(c1, c2 | self.edgesBetween(c1, c2) -> size() = 0)
and
--There are more than one replicated components in the 3rd layer
groups -> at(3) -> size() > 1
and
--Replicated components should be connected to either dispatchers and centralizers
groups -> at(3) -> forAll(c| c.predecessors() -> notEmpty() and c. successors() ->
notEmpty())}
```

4.36 (47, 50, 53) in CIRing

```
--Each component are connected to exactly 2 other components
{self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) -> forAll(c |
c.neighbors() -> size() = 2)
and
--There is no component having only successors
self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) -> select(c |
c.predecessors() -> size() = 0) -> size() = 0
and
--There is no component having only predecessors
self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) -> select(c |
c.successors() -> size() = 0) -> size() = 0}
```

4.37 (48, 51, 54) in HRing

```
{let components : Set(CLACSProfile::ComponentInstance)) =
self.base_Component.ownedAttribute -> select(oclIsTypeOf(CLACSProfile::ComponentInstance)) -
> select(c| componentNames -> includes(c.name)) in
--Each component among selected components are connected to exactly 2 other components
components ->forAll(c| c.neighbors(components) -> size() = 2)
and
--There is no component having only successors in the selected components
components -> select(c | c.predecessors(components) -> size() = 0) -> size() = 0
and
--There is no component having only predeccessors in the selected components
components -> select(c | c.successors(components) -> size() = 0) -> size() = 0
}
```

4.38 (49, 52, 55) in GRing

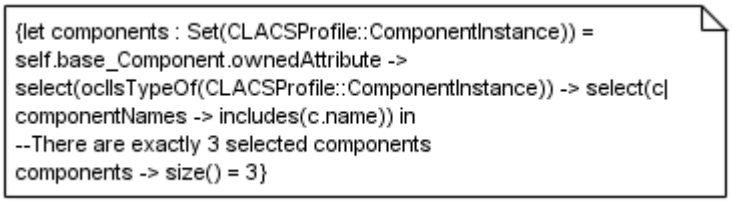
```
{let groups: OrderedSet(Set(CLACSProfile::ComponentInstance)) = componentNames -> iterate(g;
groupSet: OrderedSet(Set(CLACSProfile::ComponentInstance)) =
OrderedSet{}| let aGroup = self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->
select(c| g -> includes(c.name)) in groupSet.append(aGroup)) in
-- Each group of components should be connected to exactly 2 other groups of components
groups ->forAll(g| self.neighborsOf(g, groups) -> size() = 2)
and
--There is no group of components having only successors
groups -> select(g | self.predecessorsOf(g, groups) -> size() = 0) -> size() = 0
and
--There is no group of components having only predecessors
groups -> select(g | self.successorsOf(g, groups) -> size() = 0) -> size() = 0
}
```

4.39 (56, 57, 58) in Microkernel

```
{let groups: OrderedSet(Set(CLACSProfile::ComponentInstance)) = componentNames -> iterate(g;
groupSet: OrderedSet(Set(CLACSProfile::ComponentInstance)) =
OrderedSet{}| let aGroup = self.base_Component.ownedAttribute ->
select(oclIsTypeOf(CLACSProfile::ComponentInstance)) ->
select(c| g -> includes(c.name)) in groupSet.append(aGroup)) in
-- There are exactly 4 layers
groups ->size() = 4
and
--The 3rd layer (microkernel layer) has only 1 component
groups -> at(3) -> size() = 1
and
--The 4th layer (internal server layer) has only predecessors
self.successorsOf(groups -> at(4), groups) -> size() = 0
}
```

4.40 (59) in CIMVC

4.41 (60) in HMVC

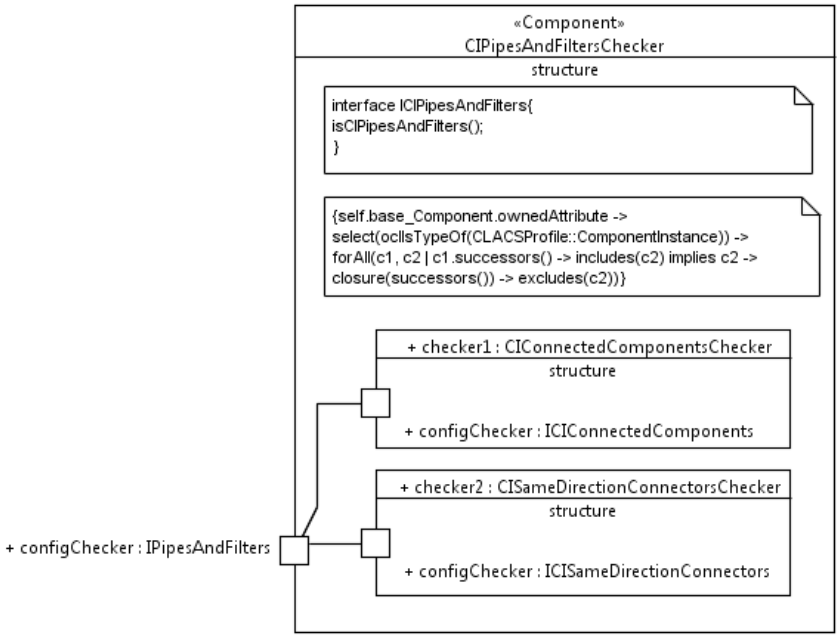


5 List of architectural patterns represented by constraint components

This section will represent the pattern catalogue presented in Section 3 but patterns are described by composing constraint components.

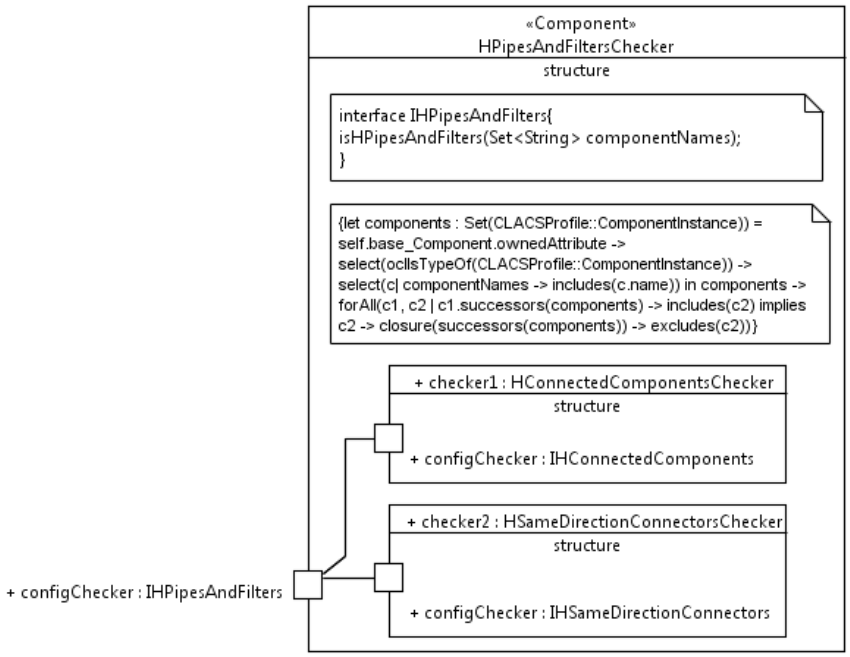
5.1 (1) CIPipesAndFiltersChecker

Variant type: Context Independent
Semantics: Pipes and Filters pattern
Configuration:



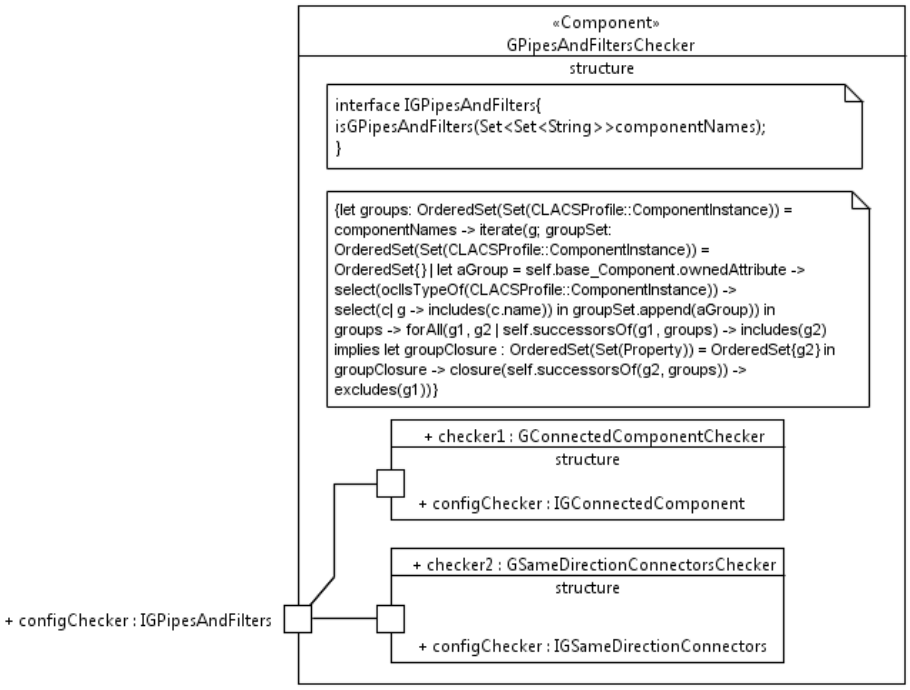
5.2 (2) HPipesAndFiltersChecker

Variant type: Hybrid
Semantics: Pipes and Filters pattern
Configuration:



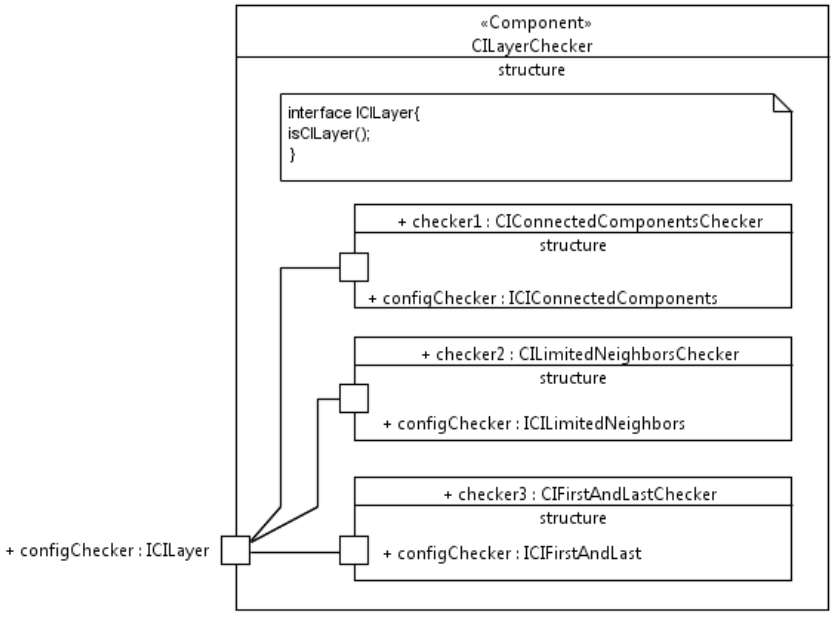
5.3 (3) GPipesAndFiltersChecker

Variant type: Group
Semantics: Pipes and Filters pattern
Configuration:



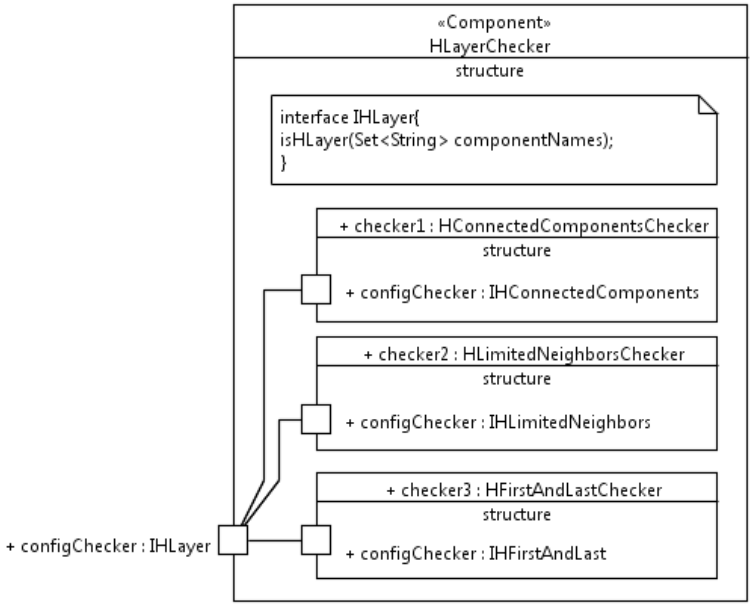
5.4 (4) CILayerChecker

Variant type: Context Independent
Semantics: Layer pattern
Configuration:



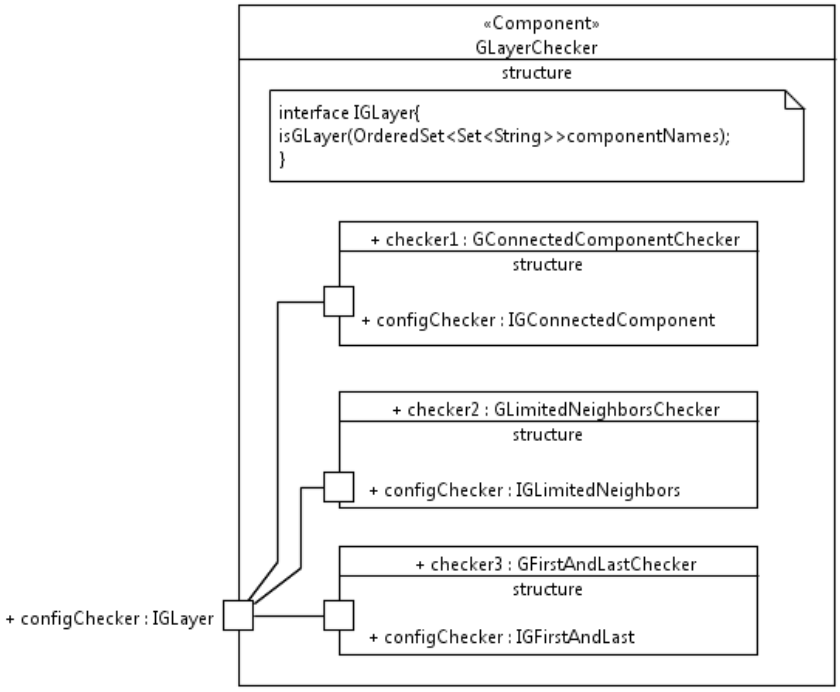
5.5 (5) HLayerChecker

Variant type: Hybrid
Semantics: Layer pattern
Configuration:



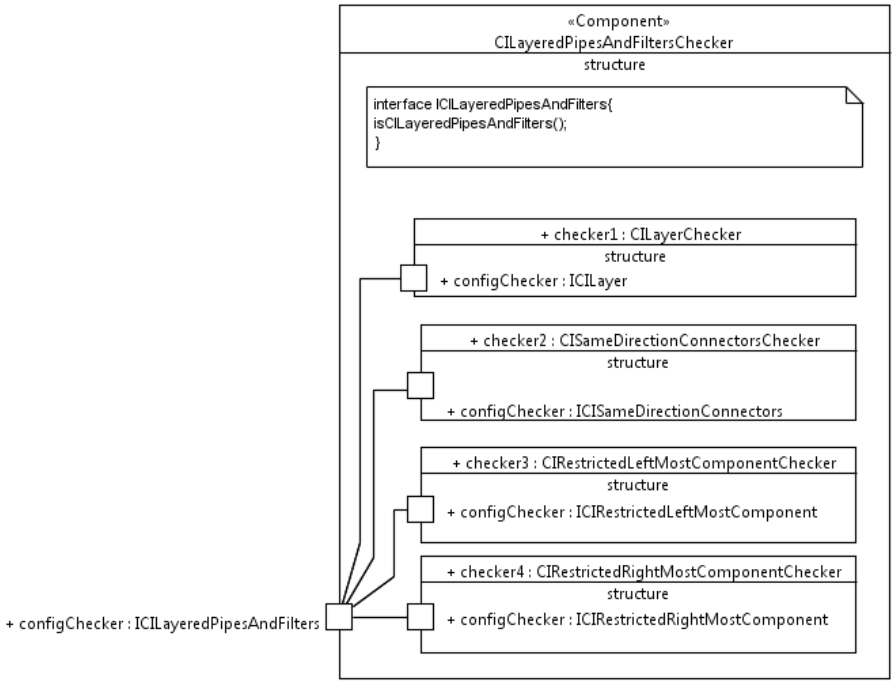
5.6 (6) GLayerChecker

Variant type: Group
Semantics: Layer pattern
Configuration:



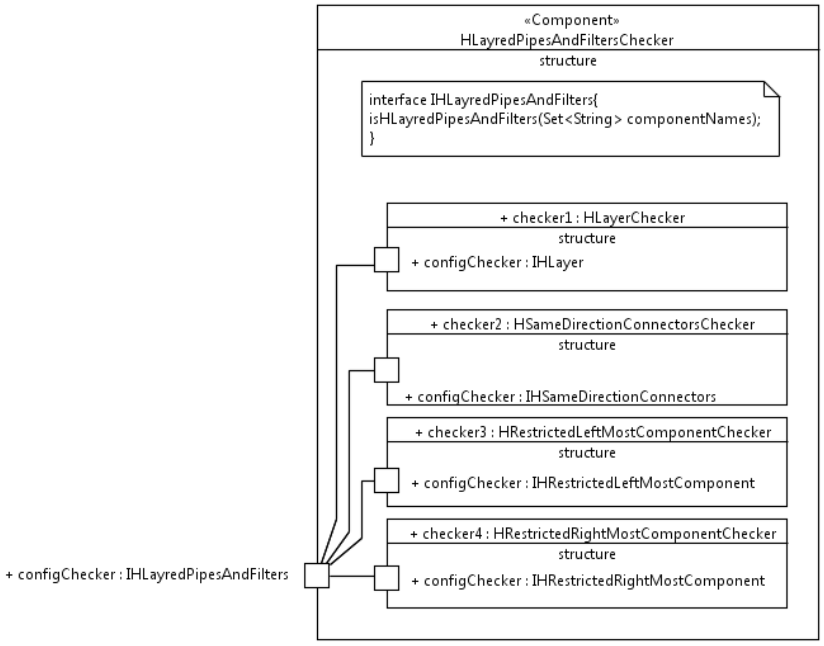
5.7 (7) CILayredPipesAndFiltersChecker

Variant type: Context Independent
Semantics: Layered Pipes and Filters pattern
Configuration:



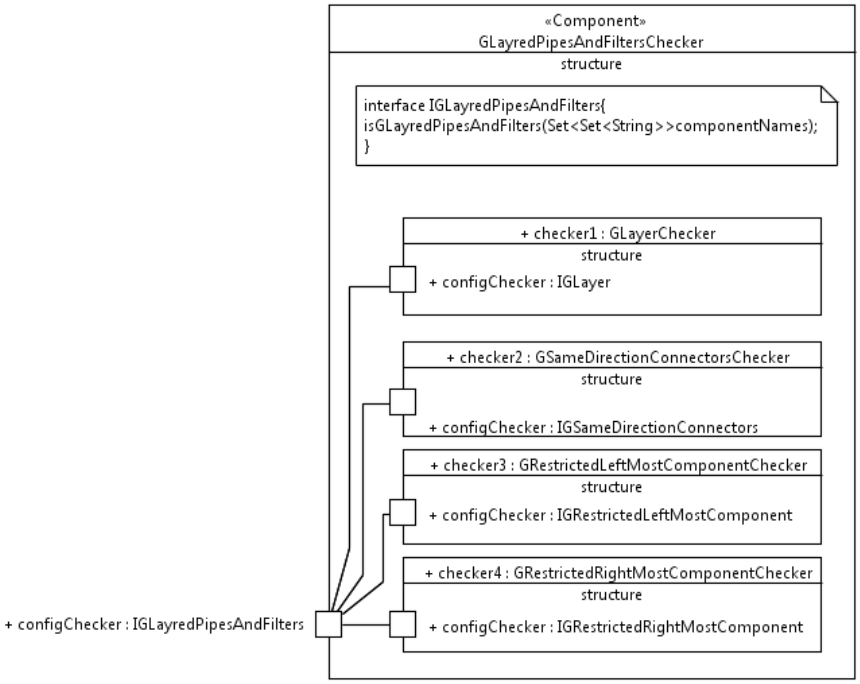
5.8 (8) HLayredPipesAndFiltersChecker

Variant type: Hybrid
Semantics: Layered Pipes and Filters pattern
Configuration:



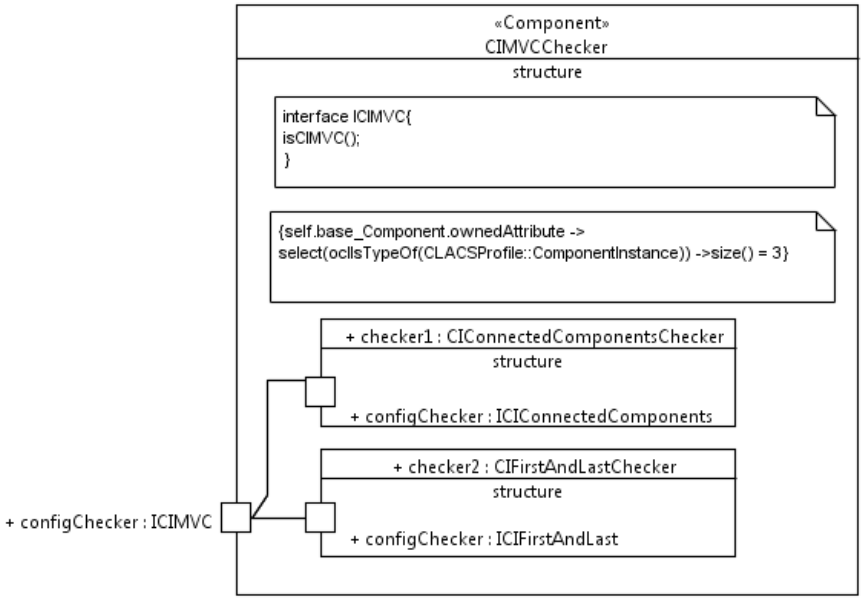
5.9 (9, 10) GLayredPipesAndFiltersChecker

Variant type: Group
Semantics: Layered Pipes and Filters pattern, N-tier pattern
Configuration:



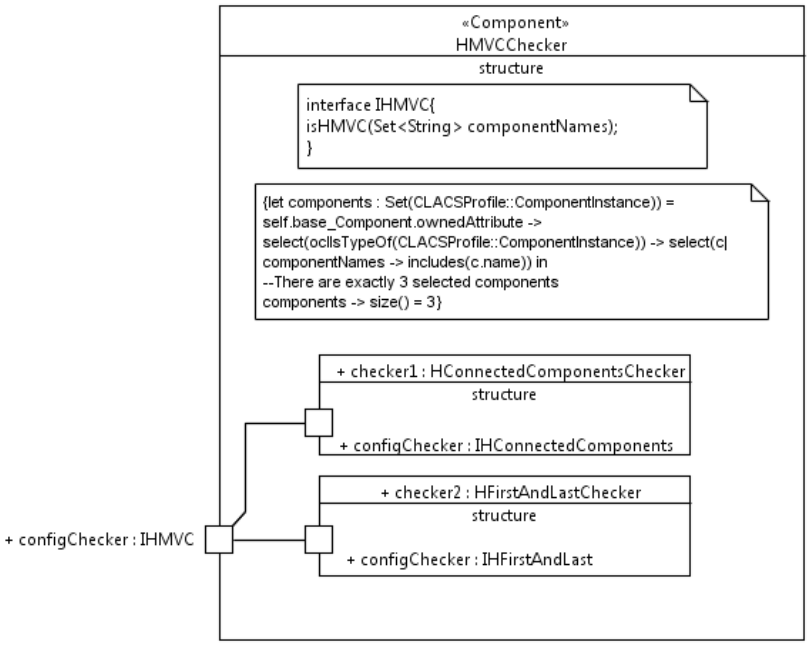
5.10 (11) CIMVCChecker

Variant type: Context Independent
Semantics: MVC pattern
Configuration:



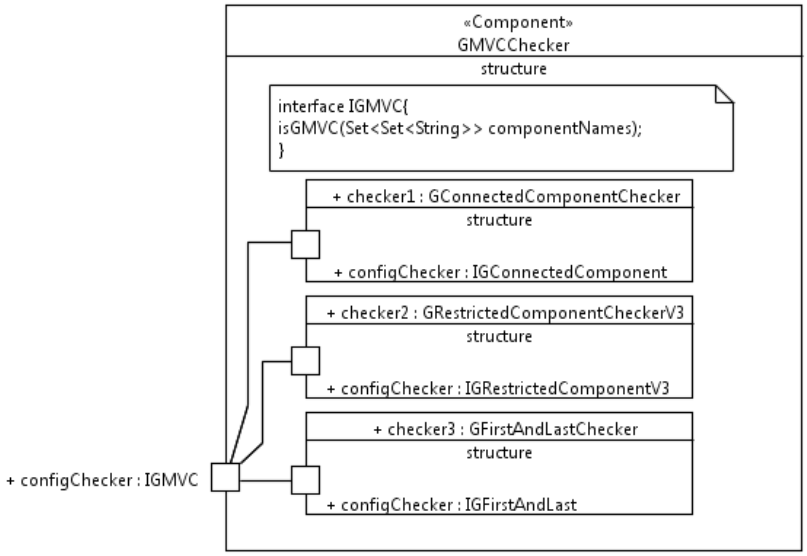
5.11 (12) HMVCChecker

Variant type: Hybrid
Semantics: MVC pattern
Configuration:



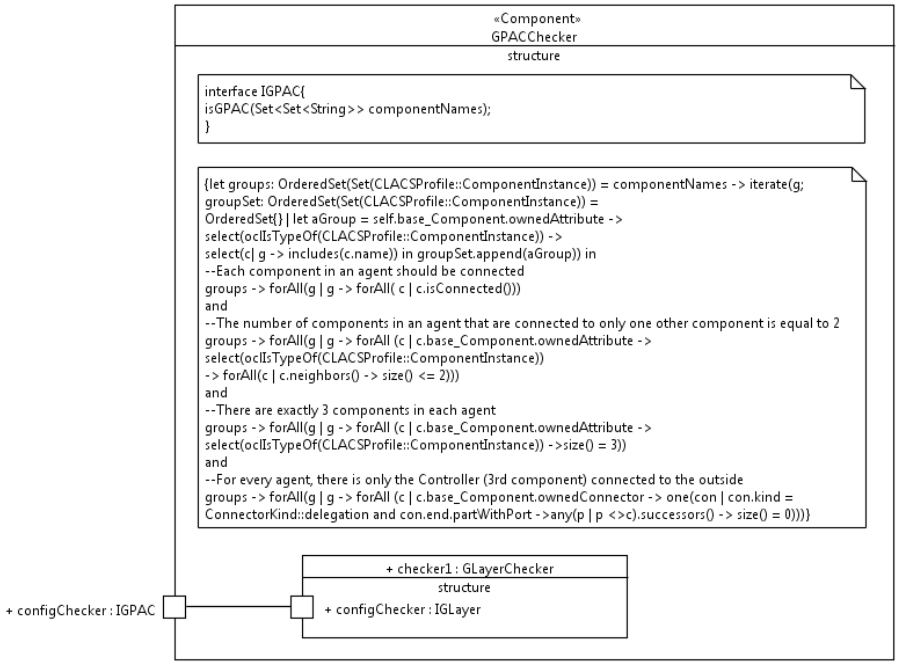
5.12 (13) GMVCChecker

Variant type: Group
Semantics: MVC pattern
Configuration:



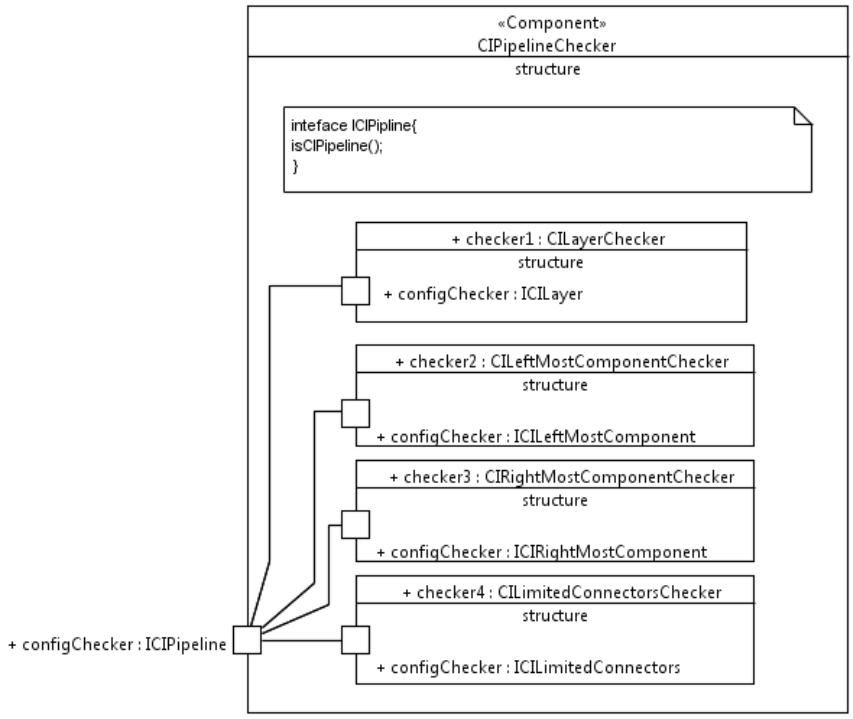
5.13 (14) GPACChecker

Variant type: Group
Semantics: PAC pattern
Configuration:



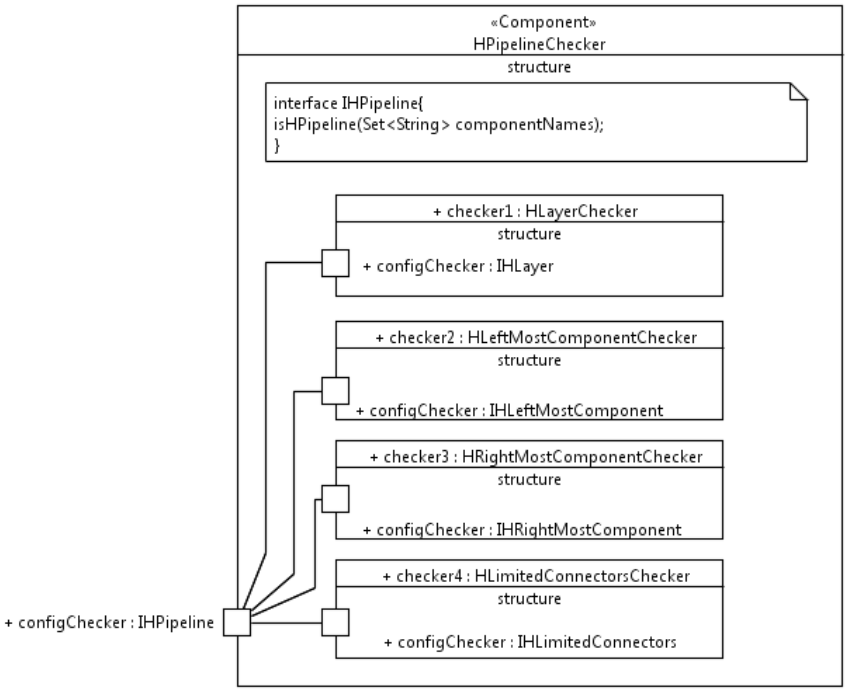
5.14 (15) CIPipelineChecker

Variant type: Context Independent
Semantics: Pipeline pattern
Configuration:



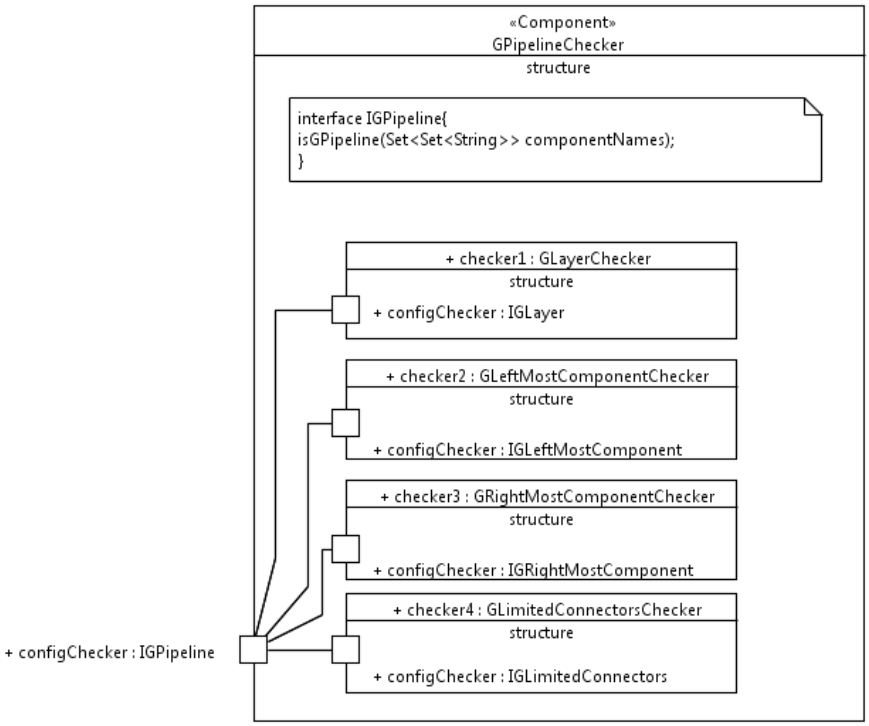
5.15 (16) HPipelineChecker

Variant type: Hybrid
Semantics: Pipeline pattern
Configuration:



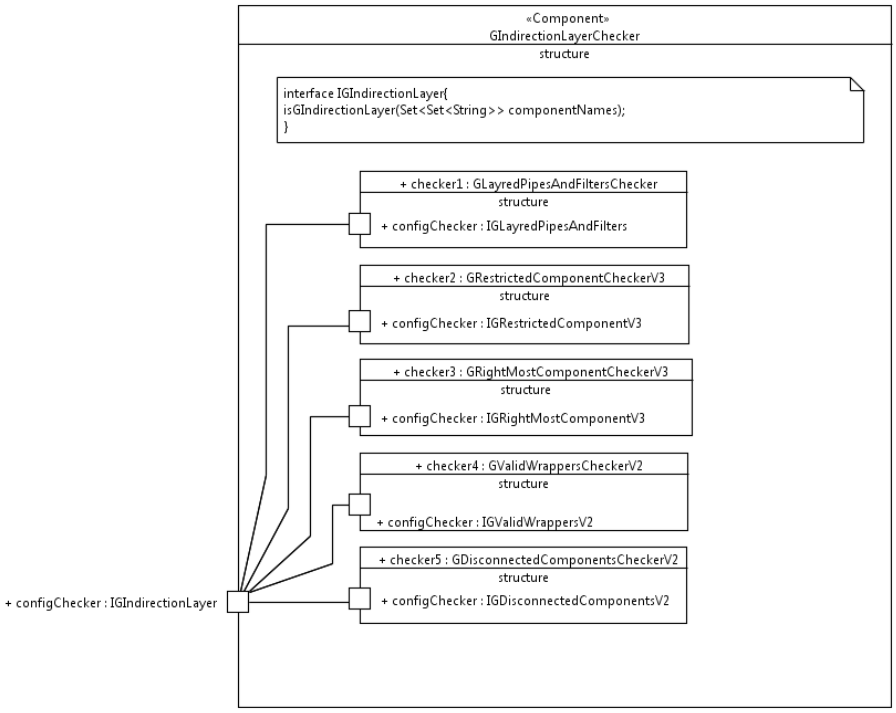
5.16 (17) GPipelineChecker

Variant type: Group
Semantics: Pipeline pattern
Configuration:



5.17 (18) GIndirectionLayerChecker

Variant type: Group
Semantics: Indirection Layer pattern
Configuration:

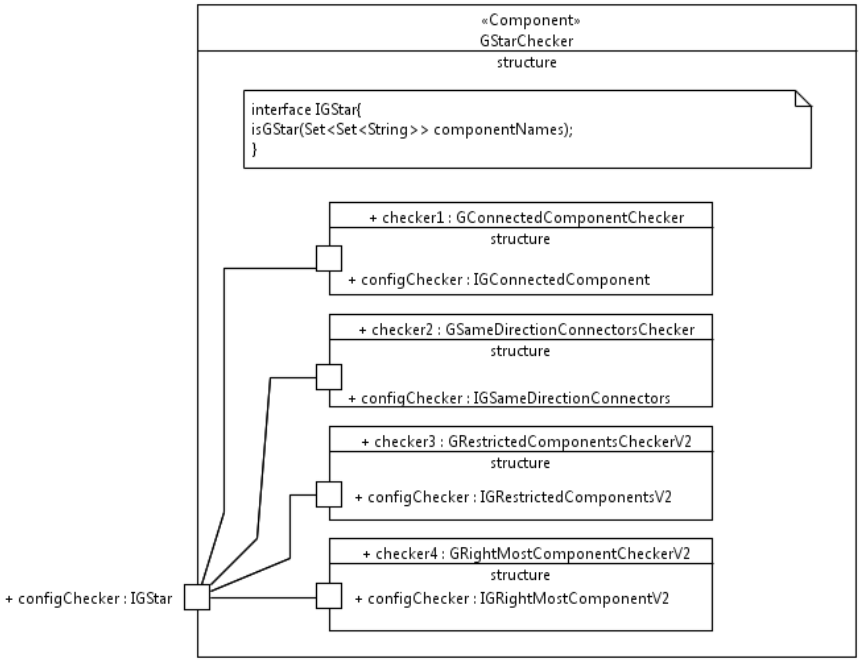


5.18 (19, 20, 21) GStarChecker

Variant type: Group

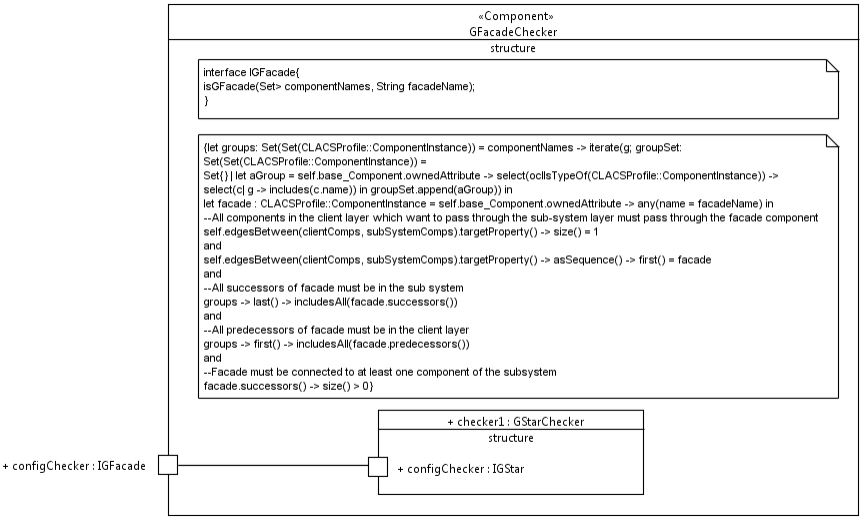
Semantics: Shared Repository pattern, Active Repository pattern, Black Board pattern

Configuration:



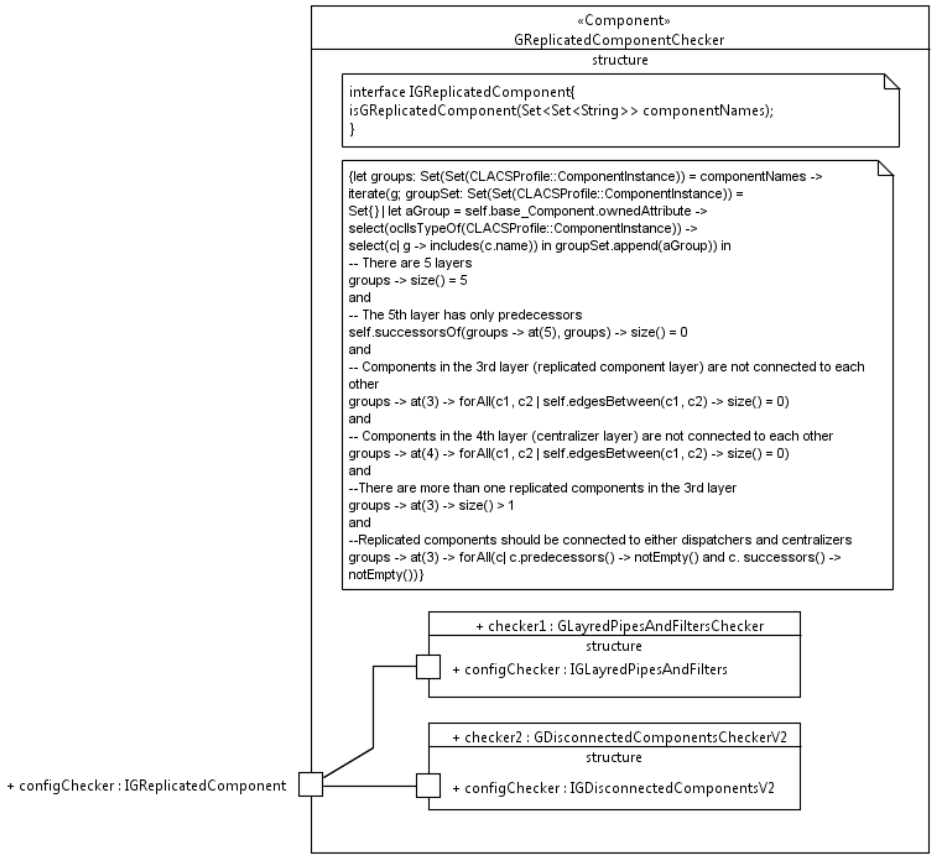
5.19 (22) GFacadeChecker

Variant type: Group
Semantics: Facade pattern
Configuration:



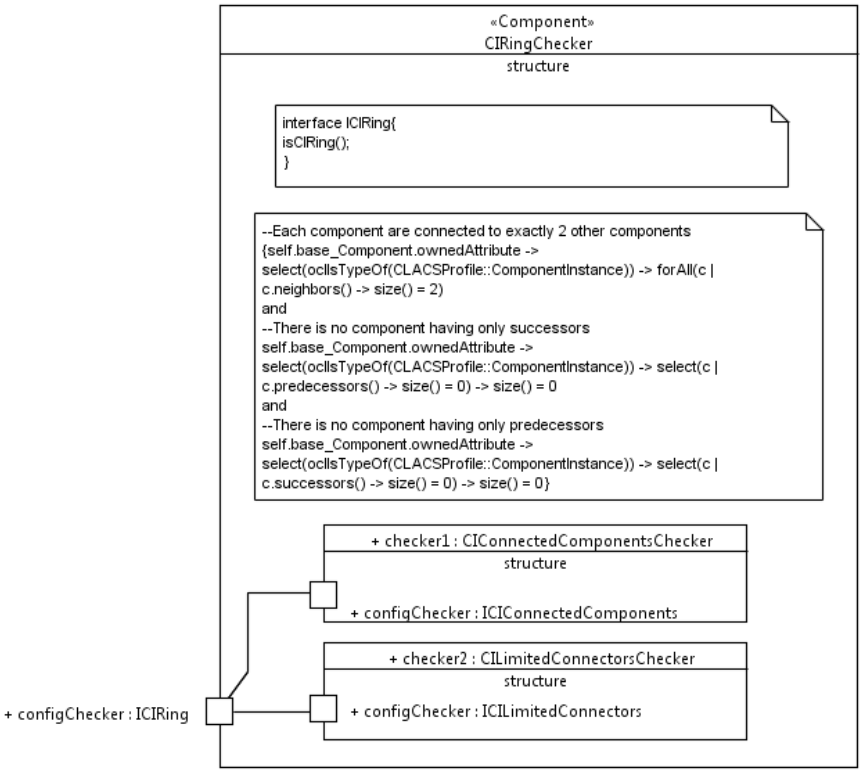
5.20 (23) GReplicatedComponentChecker

- Variant type: Group
- Semantics: Replicated Component pattern
- Configuration:



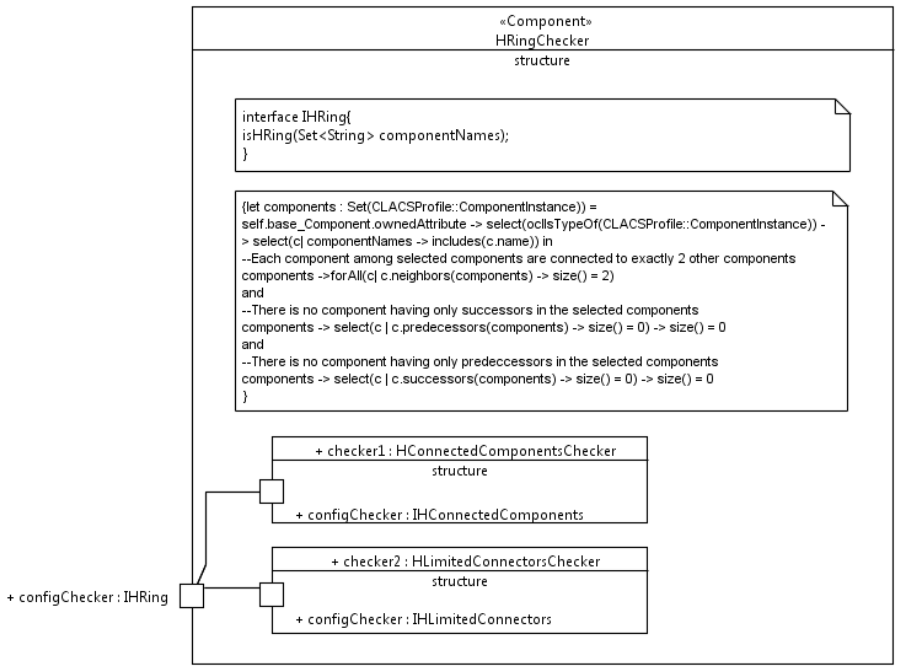
5.21 (24) CIRingChecker

Variant type: Context Independent
Semantics: Ring pattern
Configuration:



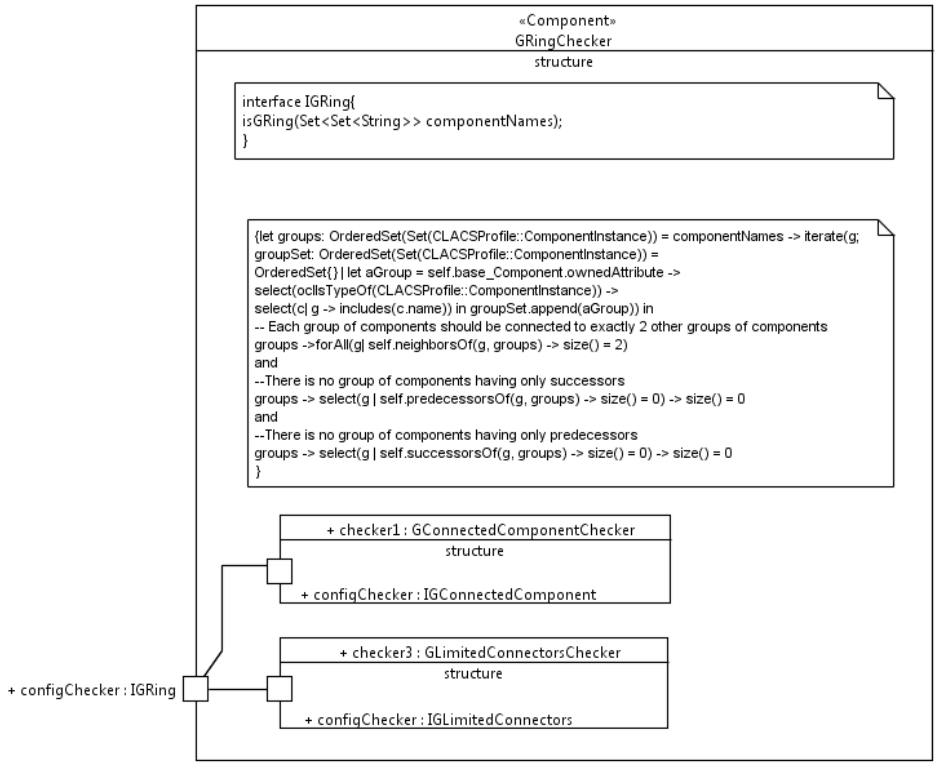
5.22 (25) HRingChecker

Variant type: Hybrid
Semantics: Ring pattern
Configuration:



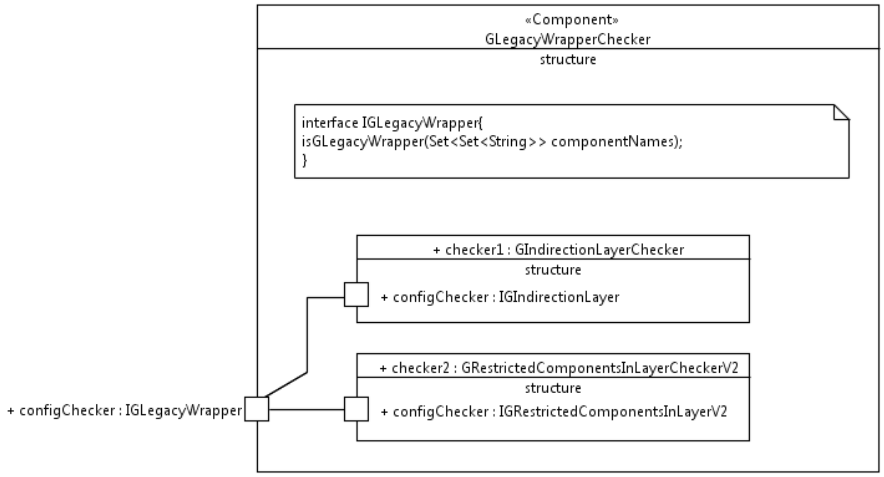
5.23 (26) GRingChecker

Variant type: Group
Semantics: Ring pattern
Configuration:



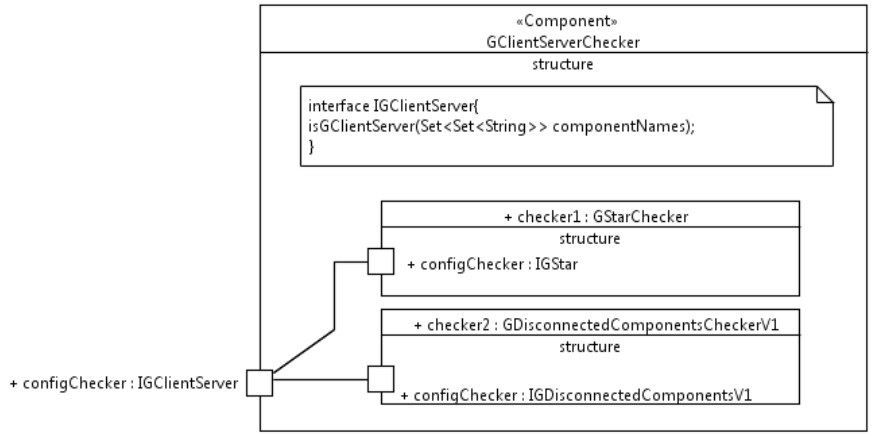
5.24 (27) GLegacyWrapperChecker

Variant type: Group
Semantics: Legacy Wrapper pattern
Configuration:



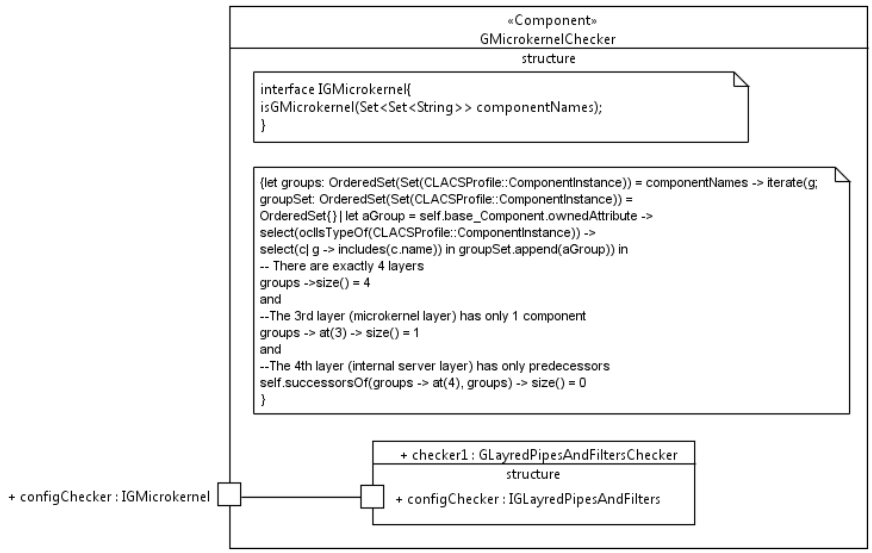
5.25 (28) GClientServerChecker

Variant type: Group
Semantics: Client Server pattern
Configuration:



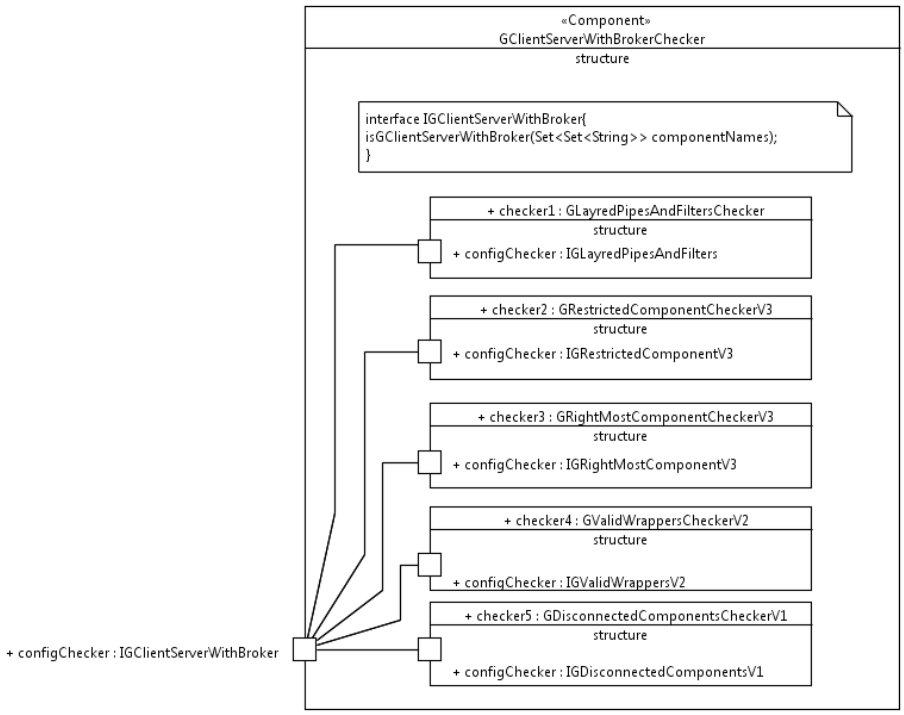
5.26 (29) GMicrokernelChecker

Variant type: Group
Semantics: Microkernel pattern
Configuration:



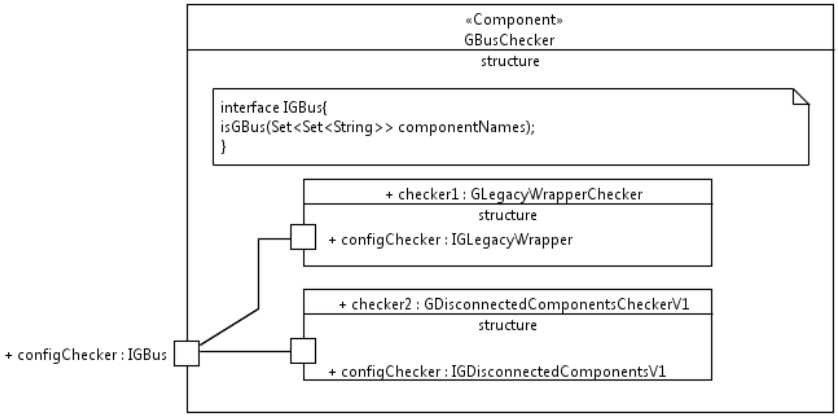
5.27 (30) GClientServerWithBrokerChecker

Variant type: Group
Semantics: Client Server With Broker pattern
Configuration:



5.28 (31) GBusChecker

Variant type: Group
Semantics: Bus pattern
Configuration:



References

[1] P. Avgeriou and U. Zdun, “Architectural patterns revisited – a pattern language,” in In 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, 2005, pp. 1–39.

[2] Buschmann, F., Meunier R., Rohnert, H., Sommerlad, P., and Stal, M, Pattern-Oriented Software Architecture - A System Of Patterns. New York: John Wiley & Sons, 1996.

[3] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. Documenting Software Architectures: Views and Beyond (2nd Edition). Addison-Wesley Professional, 2010.

[4] Erl, T.: SOA Design Patterns, Prentice Hall, 2009.